

# Approximate Analytical Performance Evaluation of a UNIX Based Multiprocessor Operating System

Stefan Greiner  
Duke University  
Dept. of Electrical Engineering  
Durham, NC 27708-0291, USA  
Box 90291  
greiner@ee.duke.edu

Gunter Bolch  
University Erlangen-Nuernberg  
IMMD IV  
Martensstrasse 1  
D – 91058 Erlangen  
bolch@informatik.uni-erlangen.de

Kishor S. Trivedi  
Center for Advanced Computing and Communication  
Duke University  
Dept. of Electrical Engineering  
Durham, NC 27707-0291  
kst@ee.duke.edu

## Abstract

In this paper we consider a non-product form queueing model of a multiprocessor operating system. In order to improve upon model solution time over a discrete-event simulation and a Markov chain based approach, we propose an extension to the shadow technique to allow for any mixture of preemptive and non preemptive priority scheduling strategy combined with class switching. This new technique can be applied to open, closed and mixed queueing networks. Our model results are validated against real measurements.

## 1 Introduction

Performance evaluation of current computer-and operating systems is very important because of the growing complexity of these systems. The aim of performance evaluation is to find the bottlenecks in the system and to optimize the system with respect to different measures of effectiveness such as throughput or response time. Performance evaluation should go hand in hand with system development but very often we do not have a real system to get system measures. Therefore we need a model of the system that describes the interesting behavior as accurately as possible. A very well known technique to describe a system is to use queueing networks. While developing performance models of a real multiprocessor operating system, we noted that due to the use of priority strategies, product form assumptions did not hold. Therefore we propose an extension of the shadow technique to allow for any mixture of preemptive and non preemptive priority scheduling strategy. We validate our analytic approximation against real measurement.

Several authors have developed analytic approximations for priority queueing networks. In [BKLC84] an approximation for the mean response time is given for the case that the network consists only of preemptive or non preemptive job classes. The shadow technique, developed by [Sevc77] considers only a pure preemptive service strategy. But they do not consider any mixture of preemptive and non preemptive service strategy together with class switching. Therefore we extend the shadow

technique to analyze queueing networks with class switching and any mixture of preemptive and non preemptive service strategy.

The rest of the paper is organized as follows. In Section 2, we discuss the basic operating system model that we analyze. In Section 3, we present a short overview of work that has already been done in this area. In 3.4.3 it is shown, how to extend the idea of [BKLC84] to have a mixture of preemptive and non preemptive service strategy but it will be seen that this very specific mixture of preemptive and non preemptive service strategy can not be applied to our model. In Section 4, we introduce the concept of chains and the shadow technique. This technique is then extended to analyze queueing networks with class switching and any kind of mixture of preemptive and non preemptive service strategy. These techniques are then applied in Section 5 to the UNIX operating system model and compared to real system measurements.

## 2 Basic Model

### 2.1 Model of a UNIX Job

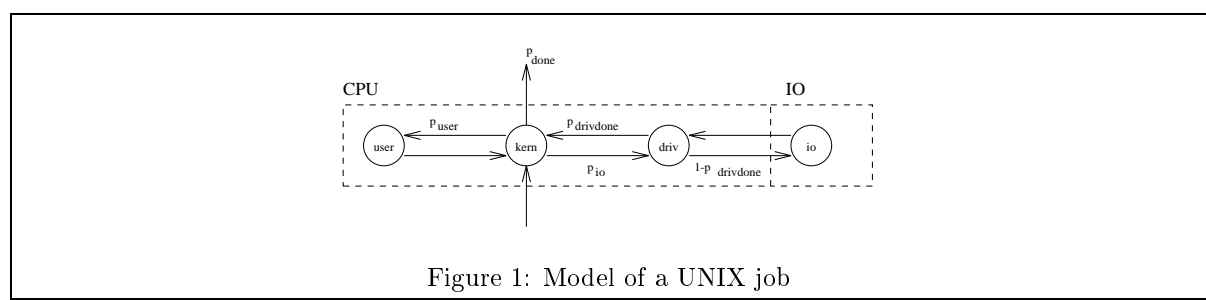


Figure 1: Model of a UNIX job

In Fig. 1 the model of a UNIX job [Jung91] during its life cycle in the system is shown. A job always starts in the kernel context *kern* and changes into the user context *user* or driver context *driv* with probability  $p_{user}$  or  $p_{io}$  respectively. A job can leave the system only in the kernel context and is replaced by a new job (closed queueing network).

### 2.2 System assumptions

- the network is closed with  $K$  jobs in the system.
- the service times are exponentially distributed.
- the peripheral device system is modeled as a load dependent station.

$s_{io}(1) = 28.00$	$s_{io}(6) = 12.444$
$s_{io}(2) = 18.667$	$s_{io}(7) = 12.000$
$s_{io}(3) = 15.555$	$s_{io}(8) = 11.667$
$s_{io}(4) = 14.000$	$s_{io}(9) = 11.407$
$s_{io}(5) = 13.067$	$s_{io}(10) = 11.200$

- the system has three job classes *user*, *kern*, *driv* with the priority order  $driv > kern > user$ .
- jobs in the context *user* can always be preempted by *driv* and *kern* jobs. Other preemptions are not possible.
- class switching is allowed.

- the system is in steady state.
- APU's (coprocessors) are used to support the CPU.
- the base model parameters are:

$K = 10$	$p_{io} = 0.05$	$s_{user} = 0.25...20.0$
$\#Apu = 2$	$p_{done} = 0.01/0.005$	$s_{kern} = 1.0$
	$p_{drivdone} = 0.4$	$s_{driv} = 0.5$

- a state in the transition probability matrix (or network routing matrix) is described as a pair  
(node number, class number)

node number 1 : CPU  
node number 2 : IO  
class number 1 : driv  
class number 2 : kern  
class number 3 : user

and the transitions between the states take place with the following probabilities

	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)
(1,1)	0	$p_{drivdone}$	0	$1 - p_{drivdone}$	0	0
(1,2)	$p_{io}$	$p_{done}$	$p_{user}$	0	0	0
(1,3)	0	1	0	0	0	0
(2,1)	1	0	0	0	0	0
(2,2)	0	0	0	0	0	0
(2,3)	0	0	0	0	0	0

### 2.3 The Master Slave Model

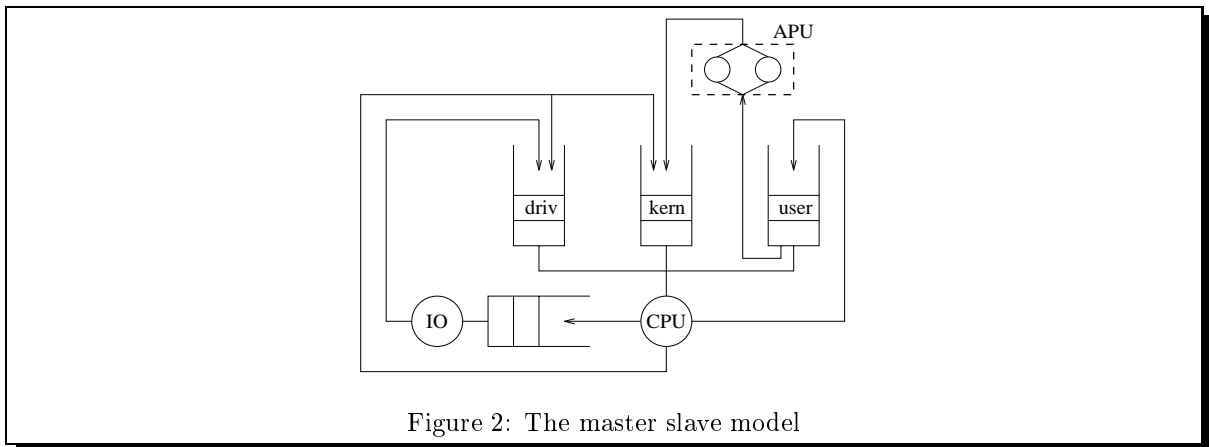


Figure 2: The master slave model

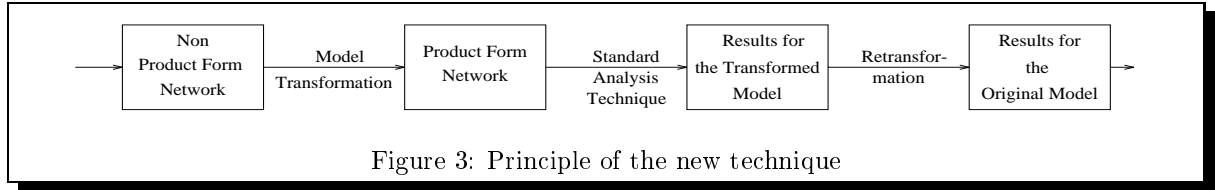
The model shown in Fig. 2 is called master slave model. In an abstract way we have a closed queueing network with three job classes where class switching is allowed. At the CPU, we have a mixture of a preemptive- and non preemptive service strategy. This means that *user* jobs can always be preempted by *kern* or *driv* jobs. Other preemptions are not possible in the system. If the CPU is free and *kern* and *driv* jobs are in the queue then *driv* jobs have higher priority. A condition for this system is, that at any time at most one job can be active at the CPU. Furthermore we have two so called APU's (coprocessors) that work together with the main CPU. We assume that only jobs in the context *user* can be served at the APU's. If APU and CPU are both free and a user job is in the queue, then it is decided randomly on which processor (CPU or APU) the job is served.

## 2.4 Difficulties in Solving this Model

The presented operating system model has the following difficulties :

- different job classes have different priorities.
- a mixed priority strategy is used in the system so that some job classes can be preempted and others can not.
- class switching is allowed.

For some of these problems separate solutions exist in the literature. The goal here is to combine these separate solutions into a new technique and to extend this technique to fulfill the needed requirements. The principle of this new technique is shown in Fig. 3:



We do not extend any existing standard analysis technique to solve a system but we change the model of the system so that it can be solved with a standard analysis technique.

## 3 Queueing Networks without Class Switching

### 3.1 Notation

- $N$  : number of nodes in the network.  
 $R$  : number of job classes.  
 $\underline{k}$  : population vector  $(k_1, k_2, \dots, k_R)$  where  $k_i$  is the number of jobs in class  $i$  ( $1 \leq i \leq R$ ) in the network.  
 $K$  : number of jobs in the closed network with

$$K = \sum_{i=1}^R k_i.$$

- $(\underline{k} - 1_r)$  : population vector with one class  $r$  job removed.  
 $\mu_{ir}$  : service rate of a class  $r$  job at node  $i$ .  
 $s_{ir} = \frac{1}{\mu_{ir}}$  : service time of a class  $r$  job at node  $i$ .  
 $\lambda_{ir}$  : mean arrival rate of a class  $r$  job at node  $i$ .  
 $m_i$  : number of servers at node  $i$ .  
 $p_{ir,js}$  : probability that a class  $r$  job moves to class  $s$  at node  $j$  after completing service at node  $i$ .  
 $e_{ir}$  : mean number of visits of a class  $r$  job at node  $i$ .

$$e_{ir} = \sum_{j=1}^N \sum_{s=1}^R e_{js} \cdot p_{js,ir}$$

- $\rho_{ir}$  : utilization of node  $i$  by class  $r$  jobs  $\rho_{ir} = \frac{\lambda_{ir}}{m_i \cdot \mu_{ir}}$ .

- $\bar{w}_{ir}$  : mean waiting time of a class  $r$  job at node  $i$ .
- $\bar{t}_{ir}$  : response time of a class  $r$  job at node  $i$ .
- $\bar{k}_{ir}$  : mean number of class  $r$  jobs at node  $i$ .
- $\bar{q}_{ir}$  : mean queue length of class  $r$  jobs at node  $i$ .

### 3.2 Description

If class switching is not allowed in the network then the number of jobs in a class is always a constant. For this type of network both exact and approximate analysis techniques are known. Many of these techniques are implemented in the tool PEPSY-QNS (**P**erformance **E**valuation and **P**rediction **S**ystem for **Q**ueueing **N**etwork**S**) at the Institute for mathematical machines and dataprocessing IV Erlangen Nuernberg [Bolc89, Kirs94]. The rest of the paper is based on the well known mean value analysis (MVA) [ReLa80] as standard analysis technique but our new technique is not only restricted to this analysis technique.

### 3.3 Mean Value Analysis

The MVA was developed by Reiser and Lavenberg [ReLa80] and can be used for the exact solution of queueing networks which consist of the following types of nodes (so called product form nodes) :

- Type 1 : M / M / m - FCFS
- Type 2 : M / G / 1 - PS (RR)
- Type 3 : M / G /  $\infty$  (Infinite Server)
- Type 4 : M / G / 1 - LCFS PR

The following two laws are fundamental to this analysis technique :

- Little's law :  $\bar{k} = \lambda \cdot \bar{t}$
- Arrival theorem by Reiser and Lavenberg :

$$\bar{t}_i(K) = \frac{1}{\mu_i} \cdot (1 + \bar{k}_i(K - 1))$$

The following equations are central for the MVA and can be derived from the arrival theorem. All further modifications are based on these equations.

$$\bar{t}_{ir}(\underline{k}) = \begin{cases} \frac{1}{\mu_{ir}} \cdot \left( 1 + \sum_{s=1}^R \bar{k}_{is}(\underline{k} - 1_r) \right) & \text{Type-1,2,4} \\ & m_i = 1 \\ \frac{1}{\mu_{ir} \cdot m_i} \cdot \left( 1 + \sum_{s=1}^R \bar{k}_{is}(\underline{k} - 1_r) + \right. \\ \quad \left. \sum_{j=0}^{m_i-2} (m_i - j - 1) \cdot p_i(j | \underline{k} - 1_r) \right) & \text{Type-1} \\ & (m_i > 1) \\ \frac{1}{\mu_{ir}} & \text{Type-3} \end{cases}$$

### 3.4 Priorities

If priority scheduling is used in the network then the product form assumptions are no longer valid. Therefore we need an approximate solution. We assume that the priority classes are ordered linearly where class 1 has the highest priority. In the derivation of the equations at first only one node is considered and the results are then integrated into the network using the mean value analysis technique.

#### 3.4.1 Preemptive Resume (PR)

The characteristic of this strategy is that as soon as a high priority job arrives in the system this job is served if no other job with higher priority is in service at the moment. If a job with lower priority is served when the high priority job arrives then this low priority job is preempted. If all jobs with higher priority are served then the preempted job is served again at the point where it was preempted. In this case [BKLC84] show, that the mean response time of a job consists of

- the service time

$$\frac{1}{\mu_r}$$

- the time a job has to wait until all jobs with higher or the same priority are served

$$\sum_{s=1}^r \frac{\bar{k}_s}{\mu_s}$$

- the time for serving all jobs with a higher priority that arrive during the response time  $\bar{t}_r$ .

$$\sum_{s=1}^{r-1} \frac{\bar{t}_r \cdot \lambda_s}{\mu_s}$$

For the priority classes  $r$  ( $1 \leq r \leq R$ ) we get [BKLC84]

$$\bar{t}_r = \sum_{s=1}^r \frac{\bar{k}_s}{\mu_s} + \sum_{s=1}^{r-1} \frac{\bar{t}_r \cdot \lambda_s}{\mu_s} + \frac{1}{\mu_r}$$

By factorizing this term and using the arrival theorem we get for node  $i$  [BKLC84]:

$$\begin{aligned} \bar{t}_{ir}(\underline{k}) &= \frac{1}{\mu_{ir}} + \frac{\sum_{s=1}^r \frac{\bar{k}_{is}(\underline{k}-1_r)}{\mu_{is}}}{1 - \sum_{s=1}^{r-1} \tilde{\rho}_{is}} \\ \tilde{\rho}_{is} &= \rho_{is}(\underline{k} - \bar{k}_{is}) \end{aligned}$$

#### 3.4.2 Non preemptive (HOL)

Here a job with a higher priority must wait until the job that is processed at the moment is ready to leave the node even if its priority is lower than the priority of the arriving job. In this case [BKLC84] show, that the mean response time of a job consists of

- the service time

$$\frac{1}{\mu_r}$$

- the time that is needed to serve a job with higher priority that arrives during the waiting time  $(\bar{t}_r - \frac{1}{\mu_r})$

$$\sum_{s=1}^{r-1} \left( \bar{t}_r - \frac{1}{\mu_r} \right) \cdot \frac{\lambda_s}{\mu_s}$$

- the time to finish the actually served job

$$\sum_{s=1}^R \frac{\rho_s}{\mu_s}$$

- the service time for jobs with the same or higher priority that are still in the queue.

$$\sum_{s=1}^r \frac{\bar{k}_s - \rho_s}{\mu_s}$$

For the priority classes  $r$  ( $1 \leq r \leq R$ ) we get [BKLC84]

$$\bar{t}_r = \sum_{s=1}^r \frac{\bar{k}_s - \rho_s}{\mu_s} + \sum_{s=1}^R \frac{\rho_s}{\mu_s} + \sum_{s=1}^{r-1} \left( \bar{t}_r - \frac{1}{\mu_r} \right) \cdot \frac{\lambda_s}{\mu_s} + \frac{1}{\mu_r}$$

By factorizing this term and using the arrival theorem we get for node  $i$  [BKLC84]

$$\begin{aligned} \bar{t}_{ir}(\underline{k}) &= \frac{1}{\mu_{ir}} + \frac{\sum_{s=1}^r \frac{\bar{k}_{is}(\underline{k}-1_r)}{\mu_{is}} + \sum_{s=r+1}^R \frac{\rho_{is}(\underline{k}-1_r)}{\mu_{is}}}{1 - \sum_{s=1}^{r-1} \tilde{\rho}_{is}} \\ \tilde{\rho}_{is} &= \rho_{is}(\underline{k} - \bar{k}_{is}) \end{aligned}$$

If we use these equations instead of the original  $\bar{t}_{ir}$  in the MVA we get an approximate technique for analyzing priority queueing networks with a PR or HOL strategy. These equations are well suited for utilizations lower than 0.7. For higher utilizations, the accuracy of these equations is not very high for class  $r > 1$  jobs but for class 1 jobs the results are very good.

### 3.4.3 Combination of PR and HOL

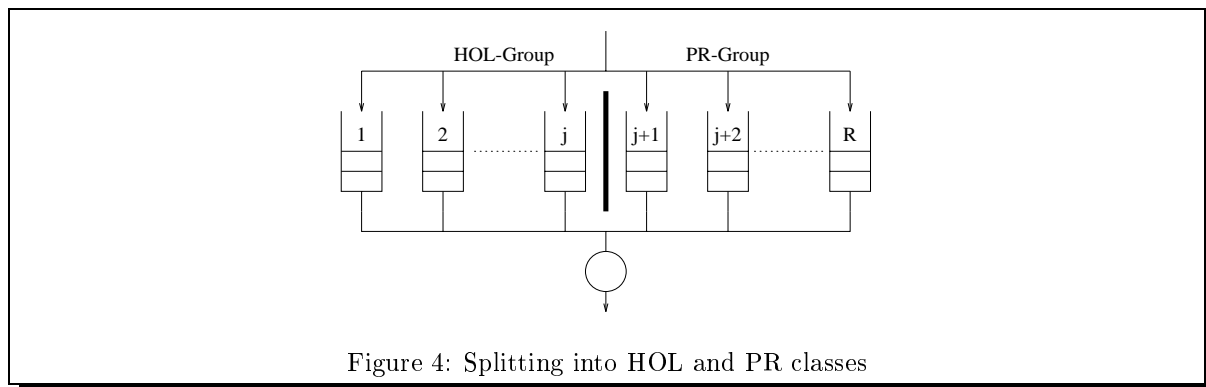


Figure 4: Splitting into HOL and PR classes

Let us consider now at a mixed strategy like the one in the UNIX-multiprocessor models of [Jung91]. That means that some job classes are of HOL type and some job classes are of PR type. It is assumed that HOL-jobs have always higher priority than PR jobs (Fig. 4). This means that we split the jobs in the two classes HOL and PR with the following assumptions :

- an arriving HOL job can not preempt a HOL job that is still in service, but it can preempt a PR job that is still in service. Therefore the response time of a HOL job is not influenced by the PR jobs

$$\bar{t}_r = \sum_{s=1}^r \frac{\bar{k}_s - \rho_s}{\mu_s} + \sum_{s=1}^j \frac{\rho_s}{\mu_s} + \sum_{s=1}^{r-1} \left( \bar{t}_r - \frac{1}{\mu_r} \right) \cdot \frac{\lambda_s}{\mu_s} + \frac{1}{\mu_r}$$

By factorizing this term we get for node  $i$

$$\begin{aligned} \bar{t}_{ir}(\underline{k}) &= \frac{1}{\mu_{ir}} + \frac{\sum_{s=1}^r \frac{\bar{k}_{is}(\underline{k}-1_r)}{\mu_{is}} + \sum_{s=r+1}^j \frac{\rho_{is}(\underline{k}-1_r)}{\mu_{is}}}{1 - \sum_{s=1}^{r-1} \tilde{\rho}_{is}} \\ \tilde{\rho}_{is} &= \rho_{is}(\underline{k} - \bar{k}_{is}) \end{aligned}$$

- a PR-job can always be preempted by a job with higher priority. For priority class  $r$  we therefore get

$$\bar{t}_r = \sum_{s=1}^r \frac{\bar{k}_s}{\mu_s} + \sum_{s=1}^{r-1} \frac{\bar{t}_r \cdot \lambda_s}{\mu_s} + \frac{1}{\mu_r}$$

By factorizing this term we get for node  $i$

$$\begin{aligned} \bar{t}_{ir}(\underline{k}) &= \frac{\frac{1}{\mu_{ir}} + \sum_{s=1}^r \frac{\bar{k}_{is}(\underline{k}-1_r)}{\mu_{is}}}{1 - \sum_{s=1}^{r-1} \tilde{\rho}_{is}} \\ \tilde{\rho}_{is} &= \rho_{is}(\underline{k} - \bar{k}_{is}) \end{aligned}$$

If these equations for  $t_{ir}(\underline{k})$  are used in the MVA instead of  $\bar{t}_{ir}$  then we get an approximate MVA for analyzing closed queueing networks with a mixed priority strategy.

## 4 Queueing Networks with Class Switching

### 4.1 MVA for Queueing Networks with Class Switching

An extension of the MVA to deal with networks with class switching is very easy if we have the concept of chains [ReLa80] and we get an algorithm that is isomorphic to the original MVA algorithm [BrBa80]. Because of the switch from classes to chains it is necessary to transfer class measures into chain measures [BrBa80].

- number of visits.  
In a chain a job can reach different states  $(i, j)$  where  
 $i$  = node number  
 $j$  = class number

$$e_{iq}^* = \frac{\sum_{r \in \pi_q} e_{ir}}{\sum_{r \in \pi_q} e_{1r}}$$

- the number of jobs per class is constant in a chain but within the chain the jobs can change their class. If therefore a job of chain  $q$  visits node  $i$  it is impossible to know of which class that job is because we look at a chain as a homogeneous construct. If we make the transformation

$$\text{class} \quad \Longrightarrow \quad \text{chain}$$

the information about the class of a job is lost. Because of the fact that different job classes have different service times, this missing information is exchanged by the scale factor  $\alpha$ . For the service rate in a chain one gets

$$s_{iq} = \frac{1}{\mu_{iq}} = \sum_{r \in \pi_q} s_{ir} \cdot \alpha_{ir} \quad \alpha_{ir} = \frac{e_{ir}}{\sum_{s \in \pi_q} e_{is}}$$

The modified MVA can now be described as follows [BrBa80]

1. calculate the number of visits  $e_{ir}$  in the original network.
2. partition the transition probability matrix  $P$  in chains  $\pi_1 \dots \pi_U$  and calculate the number of jobs  $k_i^*$  in each chain.
3. calculate the number of visits  $e_{iq}^*$  for each chain.
4. calculate the scale factors  $\alpha_{ir}$ .
5. calculate the service times  $s_{iq}$  for each chain.
6. calculate the performance parameters for each chain with the help of the extended MVA where we treat chains in the same way as classes in a system without class switching.
7. retransform the performance measures per chain into the equivalent performance measures per class.

$$\begin{aligned} \bar{t}_{ir}(\underline{k}_u) &= s_{ir} \cdot (1 + k_i^*(\underline{k}_u) - 1_q) \\ \lambda_{ir}(\underline{k}_u) &= \alpha_{ir} \cdot \lambda_{iq}^*(\underline{k}_u) \quad r \in \pi_q \\ \rho_{ir}(\underline{k}_u) &= s_{ir} \cdot \lambda_{ir}(k_u) \end{aligned}$$

Remember that

- the dimension of the population vector is  $U \leq R$ .
- in the case of  $U = R$  class switching is not possible.
- existing algorithms need not be changed but only extended in the following way :
  - determine the chains of the system.

If the number of chains equals the number of classes then class switching is not allowed and the MVA can be used with the original parameters of the network. If the number of chains is smaller than the number of classes then class switching is allowed and we have to carry out the MVA on chain values.

With the help of the chain-based technique it is possible to have an exact analysis of networks with class switching (but without priorities).

## 4.2 Networks with Class Switching and Priorities

The first opinion might be to use the equations that were presented in the case of networks without class switching (see section 3.4.1, 3.4.2), but this is not possible because

- when we make the transition from classes to chains the dimension of the population vector is normally reduced and therefore an expression like

$$\underline{k}_{ir} - 1_s$$

does not make any sense if we want to use a component  $s > U$ .

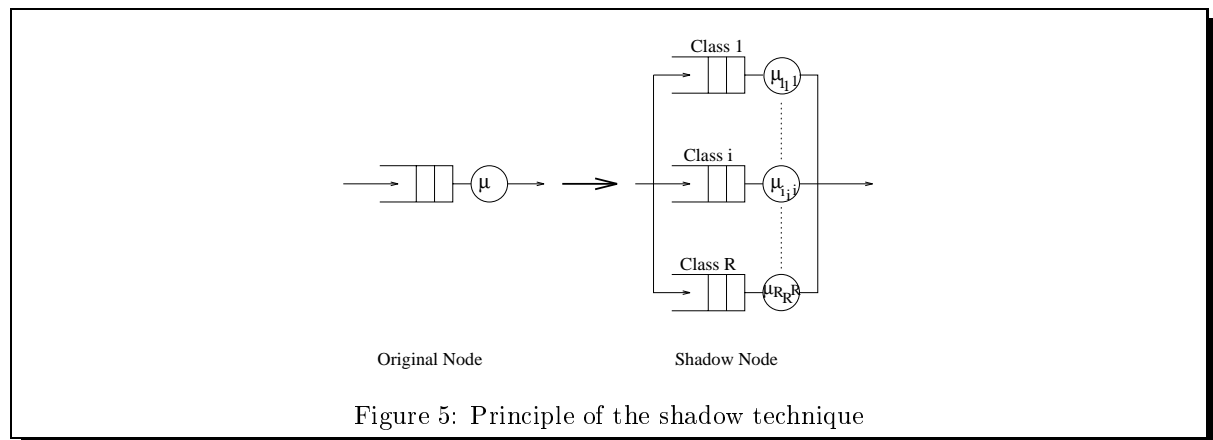
- a chain can contain jobs of different priorities and therefore a summation over all classes with higher priorities is not longer possible because we have to sum over chains.

We therefore need a technique to analyze priority networks where it is not necessary to sum over all jobs with higher priorities. For this reason the idea of shadow server is chosen. This technique is then extended to handle networks with class switching and priorities.

### 4.2.1 The Shadow Technique

This technique was introduced by [Sevc77] for networks with a pure PR-strategy. The principle is that each node in the network is split into  $R$  parallel nodes, one for each priority class (see Fig. 5). The problem then is that in this extended node jobs can run in parallel whereas this is not possible in the original node. To deal with this problem and to simulate the effect of preemption the service time for each job class in the shadow node is iteratively increased. The lower the priority of a job the higher is the service time for the job. After all iterations we have

$$s_{i,j} \geq s_{i,j}$$



The fact that for the computation of the service rates no summation over other priority classes is necessary makes this technique suitable for priority networks with class switching. To differentiate between a shadow node and the original node a notation of the form  $(i, r)$  is used for the shadow nodes with

- $i$  : original node.
- $j$  : shadow node.
- $r$  : job class.

### 4.2.2 Shadow Algorithm

1. transform the original model into the shadow model.
2. set  $\lambda_{i_j,r} = 0$
3. iterate
  - (a) compute the utilization of each shadow node

$$\tilde{\rho}_{i_j,r} = \begin{cases} \lambda_{i_r,r} \cdot s_{i_r,r} & \text{if } j = r \\ 0 & \text{otherwise} \end{cases}$$

- (b) compute the shadow service times

$$s_{i_j,r} = \begin{cases} \frac{s_{i,r}}{r-1} & \text{if } j = r \\ 1 - \sum_{s=1}^{r-1} \tilde{\rho}_{i_s,s} & \\ 0 & \text{otherwise} \end{cases}$$

Here  $s_{i,r}$  is the original service time of a class  $r$  job at node  $i$  in the original network. For nodes with  $s_{i_j,r} = 0$  the corresponding visit ratio must be set to 0. All other visit ratios remain unchanged.

- (c) compute the performance parameters of the shadow model with the MVA, i.e. use the MVA with the  $s_{i_j,r}$  and compute the throughput parameters  $\lambda_{i_j,r}$  that are needed for the next iteration. If the  $\lambda_{i_j,r}$  differ less than  $\varepsilon$  in two successive iterations then stop the iteration. Otherwise go back to step 3a.

### 4.3 Extended Shadow Technique

Measurements on real systems and comparisons with simulation results have shown that the technique of shadow server is not very accurate. Because of this, [Kauf84] has suggested an extension of the original shadow technique. This extended method differs from the original method in the way that the expression

$$\sum_{s=1}^{r-1} \tilde{\rho}_{i_s,s}$$

in the iteration is multiplied with a correction factor. This correction factor prevents the iteratively computed service times from growing too fast. For this reason the correction factor  $\delta$  is defined as follows [Kauf84] :

$$\delta_{i_j,r} = \begin{cases} \frac{\varrho(r)[1 - \varrho(r)] + \alpha(r) \cdot \varrho(r+1)}{\varrho(r)[1 - \varrho(r)]^2 + \alpha(r) \cdot \varrho_{i_r,r}} & \text{for } r = 2..R \quad \text{and} \quad j = r \\ 0 & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} \alpha(r) &= \sum_{k=1}^{r-1} \frac{\rho_{i_j,r}}{\omega_{r,k}} \\ \omega_{r,k} &= \frac{s_{i,k}}{s_{i,r}} \\ \varrho(r) &= \sum_{k=1}^{r-1} \rho_{i_k,k} \end{aligned}$$

This correction factor  $\delta$  changes only step 3b in the original shadow algorithm

$$s_{i_j,r} = \begin{cases} \frac{s_{i,r}}{1 - \sum_{s=1}^{r-1} \frac{1}{\delta_{i_s,s}} \cdot \tilde{\rho}_{i_s,s}} & \text{if } j = r \\ 0 & \text{otherwise} \end{cases}$$

If the utilizations of the shadow nodes are very close to 1 the expression

$$1 - \sum_{s=1}^{r-1} \frac{1}{\delta_{i_s,s}} \cdot \tilde{\rho}_{i_s,s}$$

can be negative because of the approximate technique. In this case the iteration must be stopped.

#### 4.3.1 Extended Shadow Technique with Bisection

- Problem :

In the original and extended shadow method the expression

$$\sum_{s=1}^{r-1} \tilde{\rho}_{i_s,s}$$

can be very close to 1. Therefore the service time  $s_{i,r}$  becomes very big and the throughput through this node is very low. But then the value of the expression

$$\sum_{s=1}^{r-1} \tilde{\rho}_{i_s,s}$$

will be very small in the next iteration step and the shadow technique does not converge. This problem was not considered in the techniques until now. Therefore the shadow technique is extended to a new variant which leads to the shadow technique with bisection.

- Solution:

For the solution of this problem we have to take into consideration the fact that the real values for the performance parameters are somewhere between the two values where the iteration keeps back and forth. The idea now is to extend the MVA by bisection but this technique is only used when the normal analysis technique keeps back and forth between two values. For this reason the new response time is computed as the mean value of the old response time and the response time from the last iteration step. This response time is then used as the new response time for the MVA. With this technique it is possible to get out of the unstable state within a few steps and the technique converges.

#### 4.3.2 Extended Shadow Technique with Class Switching

1. determine the chains in the network
2. transform the original model into the shadow model
3. set  $\lambda_{i_j,r} = 0$
4. iterate

(a)

$$\tilde{\rho}_{i_j,r} = \begin{cases} \lambda_{i_r,r} \cdot s_{i_r,r} & \text{if } j = r \\ 0 & \text{otherwise} \end{cases}$$

(b)

$$s_{i_j,r} = \begin{cases} \frac{s_{i_r,r}}{1 - \sum_{s=1}^{r-1} \frac{1}{\delta_{i_s,s}} \tilde{\rho}_{i_s,s}} & \text{if } j = r \\ 0 & \text{otherwise} \end{cases}$$

Here  $s_{i_r}$  is the original service time of a class  $r$  job at node  $i$  in the original network. For nodes with  $s_{i_j,r} = 0$  the visit ratios must be zero. The rest of the visit ratios remain unchanged.

(c) transform the class values into chain values (to differentiate between class values and chain values the chain values will be marked with a \*).

(d) compute the performance parameters of the transformed shadow model with the MVA. If the model parameters keep back and forth between two specific values, then set

$$\bar{t}_{i_j r} = \frac{\bar{t}_{i_j r}^{old} + \bar{t}_{i_j r}^{new}}{2}$$

and compute the performance measures with this new response time.

(e) transform chain values into class values

If the  $\lambda_{i_j,r}^*$  (chain value) in two following steps differs less than  $\varepsilon$  stop the iteration. Otherwise go back to step a.

### 4.3.3 Class Switching and Shadow Technique with Mixed Priority Strategy

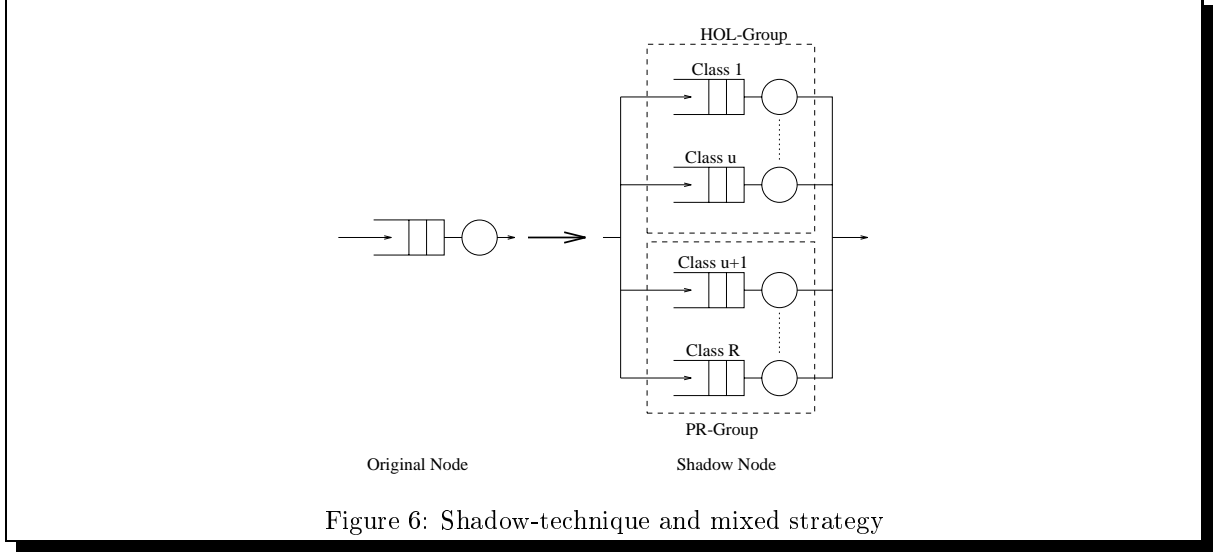
The original shadow technique [Sevc77] can only be used for networks with a pure PR-strategy but in our model of a UNIX based multiprocessor operating system a mixed priority strategy is used. For this reason the original shadow technique has to be extended to deal with this mixed priority strategy. Therefore the suggested grouping strategy (see section 3.4.3) is used. In Fig. 6 this concept is shown. Remember, that an arriving PR-job does not affect the waiting time of a HOL-job in the situation of Fig. 6 (class  $1 \dots u$  are HOL-jobs, class  $u + 1 \dots R$  are PR-jobs). On the other side a PR-job can always be preempted by an arriving HOL-job.

This situation is taken into account by modifying the correction factor  $\delta$  in the extended shadow technique [Grei93].

$$\psi_{i_j,r} = \begin{cases} \frac{\varrho(r)[1 - \varrho(r)] + \beta(r) \cdot \tilde{\varrho}(r)}{\varrho(r)[1 - \varrho(r)]^2 + \beta(r) \cdot \varrho_{i_r,r}} & \text{for } r = 2..R \quad \text{and} \quad j = r \\ 0 & \text{otherwise} \end{cases}$$

$$\beta(r) = \begin{cases} \sum_{k=1, k \neq r}^u \frac{\rho_{i_j r}}{\omega_{rk}} & \text{if } r \in \text{HOL} \\ \sum_{k=1}^{r-1} \frac{\rho_{i_j r}}{\omega_{rk}} & \text{if } r \in \text{PR} \end{cases}$$

(For the meaning of the other variables see section 4.3.4)



#### 4.3.4 Class Switching and any Mixed Priority Strategy with the Shadow Technique

Until now we assumed that the classes  $1, \dots, u$  are pure HOL-classes and the classes  $u + 1, \dots, R$  are pure PR-classes. In the derived equations we can see that in the HOL-case the summation in the correction factor is over all HOL-classes except the actual considered class. In the PR-case the summation is over all classes with higher priority. With this relation in mind it is possible to extend the equations in such a way that any mixture of PR and HOL-classes is possible. Let  $\xi_r$  be the set of jobs that can not be preempted by a class  $r$  job. In addition it is assumed that  $r \notin \xi_r$ . For the computation of the correction factor this mixed priority strategy has the consequence that

- for a HOL job class  $r$  the summation is not from 1 to  $u$  but over all priority classes  $s \in \xi_r$ .
- for a PR job class  $r$  the summation is not from 1 to  $r - 1$  but over all priority classes  $s \in \xi_r$ .

For any mixed priority strategy we get the following equations

$$s_{i_j, r} = \begin{cases} \frac{s_{i, r}}{1 - \sum_{s \in \xi_r} \frac{1}{\psi_{i_s, s}} \cdot \tilde{\rho}_{i_s, s}} & \text{if } r = j \\ 0 & \text{otherwise} \end{cases}$$

$$\psi_{i_j, r} = \begin{cases} \frac{\varrho(r)[1 - \varrho(r)] + \beta(r) \cdot \tilde{\varrho}(r)}{\varrho(r)[1 - \varrho(r)]^2 + \beta(r) \cdot \varrho_{i_r, r}} & \text{if } r = j \\ 0 & \text{otherwise} \end{cases}$$

$$\beta(r) = \sum_{k \in \xi_r} \frac{\rho_{i_j, r}}{\omega_{r, k}}$$

$$\omega_{r, k} = \frac{s_{ik}}{s_{ir}}$$

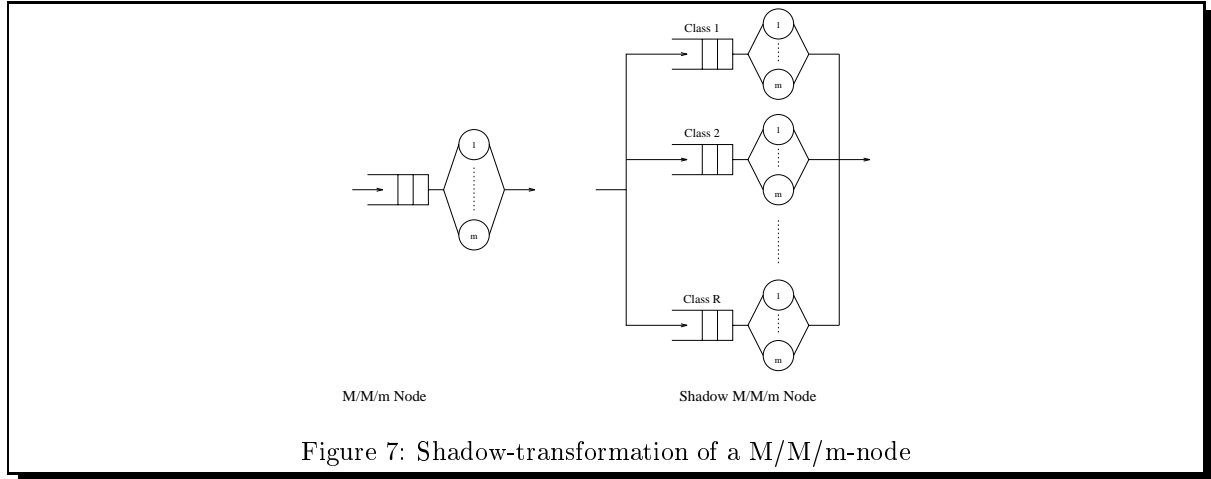
$$\varrho(r) = \sum_{k \in \xi_r} \rho_{i_k, k}$$

$$\tilde{\varrho}(r) = \sum_{k \in \{\xi_r \cup r\}} \rho_{i_k, k}$$

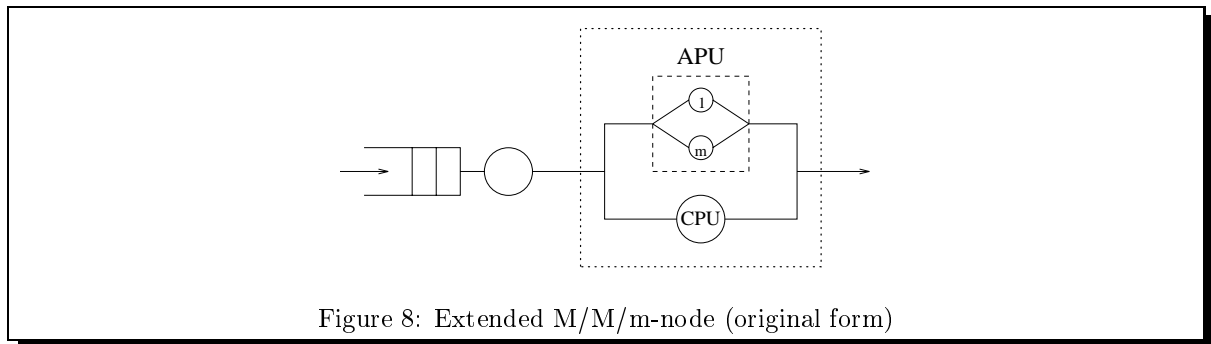
#### 4.3.5 Shadow Technique and Extended M/M/m-node

Now we introduce a new type of node - the so called extended M/M/m-node. This new node type plays a major role in the UNIX multiprocessor model.

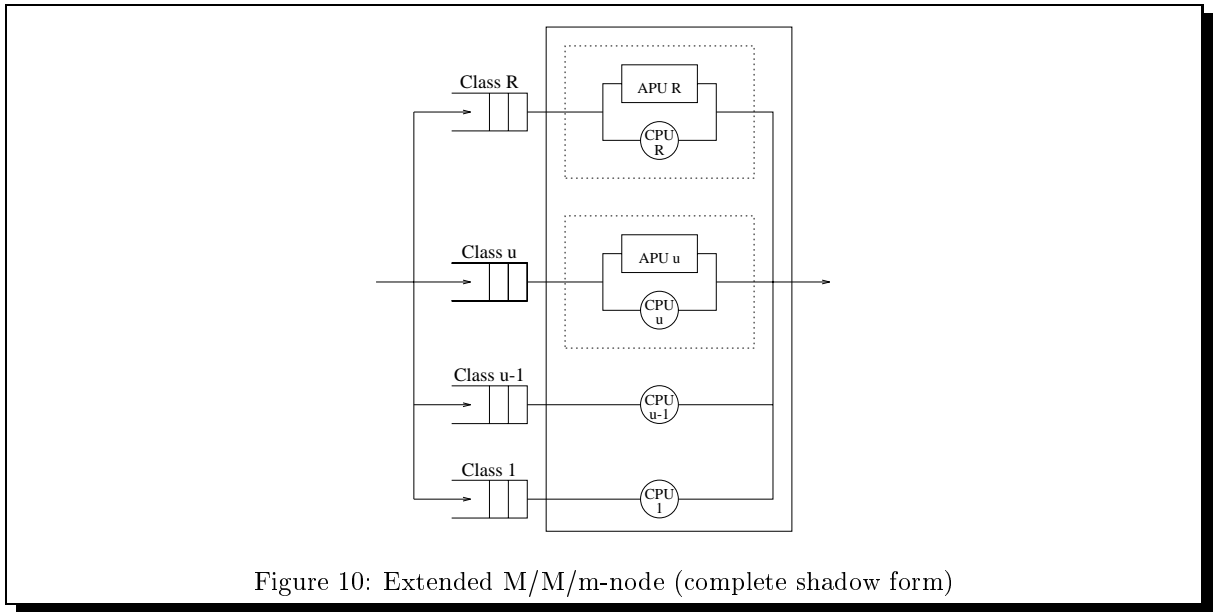
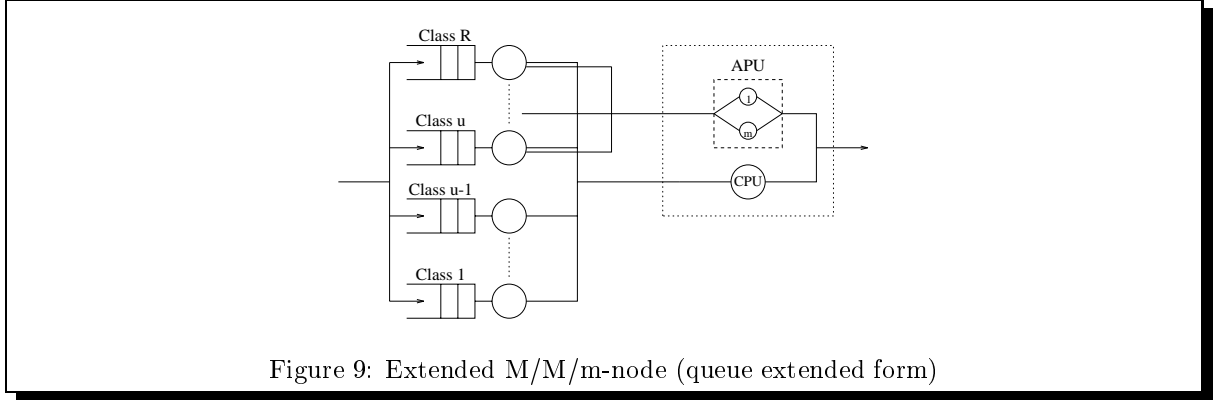
In the case of a M/M/m-node it is very easy to analyze this node, we only have to transform the M/M/m-node into its shadow form (see Fig. 7) and use the equations for M/M/m-nodes when analyzing the node with the mean value analysis in each shadow iteration. The difference between



the M/M/m-node shown in Fig. 7 and the extended M/M/m-node in Fig. 8 is that in the extended M/M/m-node we have a node with  $R$  job classes which are ordered in priority order. The CPU can be entered by all job classes while the so called APU (associate processing unit) can only be entered by the job classes  $u..R$  ( $1 \leq u \leq R$ ). For a better overview a queue is introduced for each job class.



The result of this transformation is shown in Fig. 9. If for example  $u = R = 3$  then the job classes 1, 2, 3 can enter the CPU while the APU can only be entered by job class 3. In the CPU class 3 jobs can always be preempted by class 1 and 2 jobs while in the APU class 3 jobs can not be preempted (in this situation). This means that the priority of a job depends on which node it enters. In general, preemption is also possible at the APU. To deal with this problem we have to use the idea of the shadow server again. The result is shown in Fig. 10. With this transformations done on the original network, it is possible to use the extended MVA for networks with priority strategies. Because of the fact that the shadow iterations are done separately on the CPU and APU, the service times at the different service stations are different. Therefore we get an asymmetric node. Let  $\zeta$  be the set which



contains the highest priority of the job that can enter the service station (CPU, APU). In a general case if we have a so called *generalized extended M/M/m node* [Grei93],  $\zeta$  contains normally more than two elements. In [Grei93] it is shown that in the shadow algorithm only the step for computing the mean service time  $s_{i_j,r}$  has to be changed in the following way :

$$\forall c \in \zeta \quad : \quad s_{i_j,r} = \begin{cases} \frac{s_{i_r}}{1 - \sum_{s \geq c, s \in \xi_r} \frac{1}{\psi_{i_s,s}} \cdot \tilde{\rho}_{i_s,s}} & \text{if } r = j \\ 0 & \text{otherwise} \end{cases}$$

If we apply this transformation techniques to our operating system model (see Fig. 2) then this transformed model looks as follows:

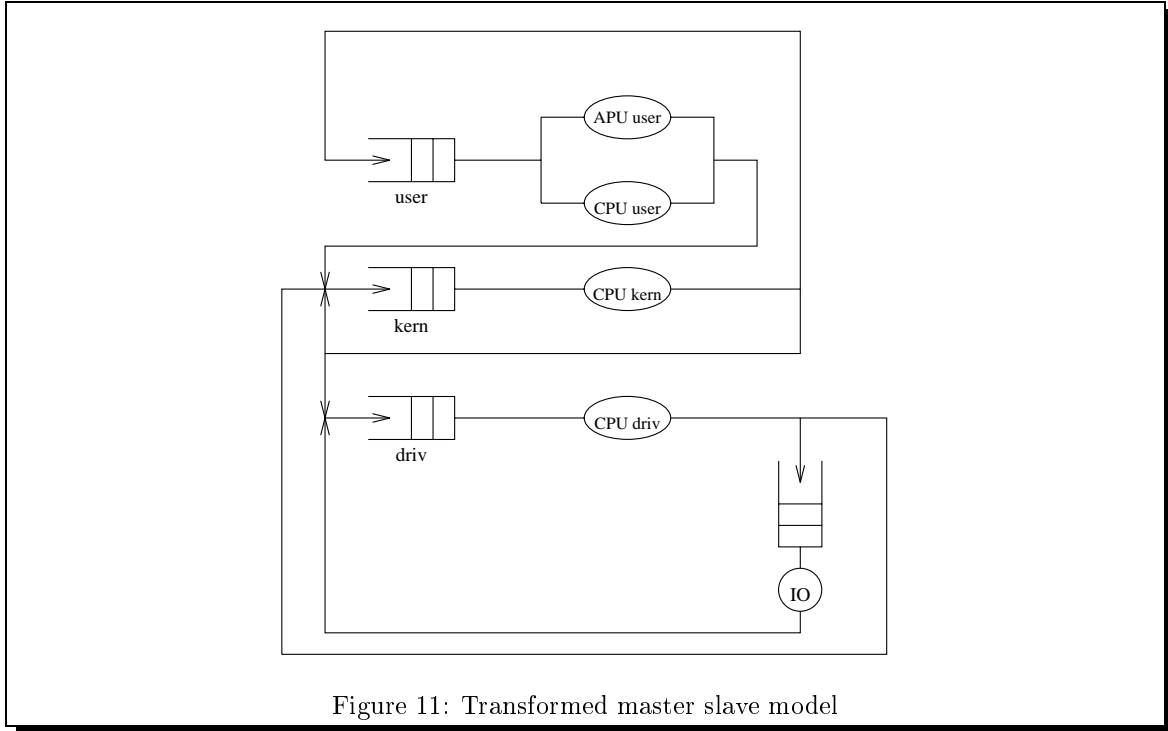


Figure 11: Transformed master slave model

This transformed model can now be analyzed, using the presented modification of the shadow technique.

## 5 Evaluation and Validation of the Techniques

The use of the presented techniques is now shown for the master slave multiprocessor operating system model (see section 2.3) . For this model the parameters were varied over a wide range. We compare the throughputs obtained from our approximation techniques with these obtained from real system measurements.

### 5.1 Variation of the Transition Probabilities

$s_{user}$	$p_{done} = 0.01$		$p_{done} = 0.005$		$p_{done} = 0.0025$	
	approx	orig	approx	orig	approx	orig.
0.25	8.963	8.963	4.481	4.481	2.240	2.241
0.50	8.933	8.930	4.466	4.465	2.233	2.232
1.0	8.842	8.841	4.420	4.420	2.210	2.210
1.5	8.678	8.678	4.337	4.337	2.168	2.168
2.0	8.390	8.368	4.191	4.180	2.094	2.089
5.0	5.120	5.182	2.548	2.580	1.271	1.287
10.0	2.642	2.867	1.314	1.427	0.655	0.712
20.0	1.372	1.510	0.660	0.751	0.329	0.375

$s_{user}$	$p_{drivdone} = 0.3$		$p_{drivdone} = 0.4$		$p_{drivdone} = 0.5$	
	approx	orig	approx	orig	approx	orig.
0.25	7.071	7.056	8.963	8.936	9.485	9.448
0.50	7.045	7.025	8.933	8.930	9.477	9.479
1.0	6.979	6.956	8.842	8.841	9.440	9.443
1.5	6.884	6.866	8.678	8.678	9.327	9.313
2.0	6.746	6.736	8.390	8.368	9.034	8.945
5.0	4.883	4.997	5.120	5.182	5.184	5.212
10.0	2.633	2.859	2.642	2.867	2.646	2.871
20.0	1.326	1.509	1.327	1.510	1.327	1.511

$s_{user}$	$p_{io} = 0.07$		$p_{io} = 0.05$		$p_{io} = 0.03$	
	approx	orig	approx	orig	approx	orig.
0.25	7.608	7.593	8.963	8.963	9.620	9.620
0.50	7.579	7.558	8.933	8.930	9.613	9.615
1.0	7.501	7.479	8.842	8.841	9.584	9.585
1.5	7.388	7.372	8.678	8.678	9.472	9.447
2.0	7.220	7.211	8.390	8.368	9.150	9.029
5.0	5.048	5.144	5.120	5.182	5.096	5.136
10.0	2.692	2.915	2.643	2.867	2.592	2.820
20.0	1.355	1.539	1.327	1.510	1.300	1.482

## 5.2 Variation of the Mean Values

$s_{user}$	$s_{kern} = 1.2$		$s_{kern} = 1.0$		$s_{kern} = 0.8$	
	approx	orig	approx	orig	approx	orig.
0.25	7.782	7.785	8.963	8.963	10.181	10.169
0.50	7.770	7.771	8.933	8.930	10.123	10.107
1.0	7.731	7.732	8.842	8.841	9.947	9.944
1.5	7.660	7.657	8.678	8.678	9.640	9.655
2.0	7.522	7.500	8.390	8.368	9.141	9.163
5.0	5.060	4.998	5.120	5.182	5.162	5.372
10.0	2.640	2.813	2.642	2.867	2.645	2.923
20.0	1.326	1.495	1.327	1.510	1.327	1.526

$s_{user}$	$S_{driv} = 0.6$		$S_{driv} = 0.5$		$S_{driv} = 0.4$	
	approx	orig	approx	orig	approx	orig.
0.25	8.887	8.890	8.963	8.963	9.040	9.036
0.50	8.857	8.858	8.933	8.930	9.009	9.001
1.0	8.766	8.773	8.842	8.841	8.918	8.908
1.5	8.601	8.616	8.678	8.678	8.756	8.740
2.0	8.308	8.315	8.390	8.368	8.471	8.421
5.0	5.098	5.170	5.120	5.182	5.146	5.194
10.0	2.641	2.864	2.643	2.867	2.645	2.871
20.0	1.327	1.509	1.327	1.510	1.327	1.511

## 5.3 Impact of the Priorities

In the following table the impact of priorities is shown in the case of the master slave model. Here  $s_{user}$  is the service time for user jobs.  $p_{io}$  is varied while  $p_{done}$  is a constant. On the right side of the table the throughput for the network without priorities (nonpre), the original values (orig) and approximate values (approx) which we got with our new approximate technique are given.

$S_{user}$	$P_{io}$	$P_{done}$	Throughput		
			nonpre	approx	orig
0.50	0.9	0.01	0.6593	0.6594	0.6593
0.50	0.01	0.01	9.0049	9.8765	9.8765
0.50	0.001	0.01	9.0897	9.9875	9.9875
1.00	0.9	0.01	0.6593	0.6594	0.6593
1.00	0.01	0.01	8.2745	9.8736	9.8716
1.00	0.001	0.01	8.3398	9.9859	9.9832
1.50	0.9	0.01	0.6593	0.6594	0.6592
1.50	0.01	0.01	7.6517	9.8194	9.7666
1.50	0.001	0.01	7.7027	9.9395	9.8983
2.00	0.9	0.01	0.6593	0.6594	0.6592
2.00	0.01	0.01	7.1078	9.5178	9.4700
2.00	0.001	0.01	7.1489	9.6326	9.4000
2.50	0.9	0.01	0.6592	0.6593	0.6592
2.50	0.01	0.01	6.6170	8.8123	8.6144
2.50	0.001	0.01	6.6485	8.8802	8.5100

As can be seen the impact of priorities can not be neglected for some parameter values.

## 6 Conclusion

With the help of the newly developed approximation technique it is possible to analyze the chosen operating system model. If we use discrete event simulation to analyze the system model, it takes about two days to get performance measures for one set of parameters. If we use the newly developed approximation technique, it takes about one second to analyze the system for a whole set of parameters and in our case the differences between the measured values and the values we got from our new technique were very small. Therefore this technique is very well suited if we want to optimize the system because many parameter sets have to be analyzed. The same techniques were also used to analyze two further operating system models called *monoprocessor model* and *associated processor model*. In the monoprocessor model no APU's are used and in the associated processor model the kernel was parallelized [Jung91]. The results for the associated processor model proved that it is worth to parallelize the system kernel by using APU's. But on the other hand it is not worth to add more than two APU's because for each additional APU the system performance increases only a little. In this sense the degree of parallelism is limited.

The real flexibility of our new technique is that it is not only restricted to the mean value analysis but can also be applied to other techniques like SCAT or Convolution. To make the model of the UNIX-operating system more realistic we can also assume an open network. To solve this kind of network with our technique, we use the so called closing method [BGJ92] to transform the open network into a closed network. This closed network can then be analyzed with our new technique. Even if we have networks with generally distributed service times, we can use our new technique. As analysis technique we can choose, for instance, the method of Marie [Mari79]. As before, the network is transformed into the shadow network and analyzed iteratively with the extended shadow technique. Between two shadow iterations the Marie method can be used to get the performance measure needed for the next shadow iteration (increasing the service times). As a last example of our new technique we can consider an open queueing network with generally distributed service times. One possibility is to use an extension of the method of Kuehn [Kueh79] as the base analysis technique in our new set up. The other possibility is to use first the closing method [Gaeb92] to get a closed network and this closed network can then be analyzed using our new technique together with the method of Marie.

Because of very good results obtained from these approximations, we plan to develop queueing models for other operating systems (like MACH, Lotus, real time operating systems) and to use this new technique to analyze the system. At the moment, we are modeling the BS2000 operating system using the new technique. One of our first results was that the assumption of "exponential distribution"

is a very good assumption for the considered operating systems. The models of UNIX and BS2000 are very similar and in [Grei95] it is shown that it is possible to give a basic model for (general purpose) operating systems. This basic model is based on a black box representation of the system. The black boxes specify the general behavior of the system and the modeler only has to specify the priority mechanism and the internal state transitions for each box. The black box representation is then transformed so that the network can be analyzed using standard analysis techniques.

## References

- [BKLC84] Bryant R.M.; Krzesinski A.E.; Lakshmi M.S.; Chandy K.M.: *The MVA Priority Approximation*, ACM Transactions on Computer Systems, Vol. 2, No. 4, pp. 335 - 359, Nov. 1984.
- [Bolc89] Bolch G.: *Leistungsbewertung von Rechensystemen*, Teubner , 1989.
- [BrBa80] Bruell S.C. , Balbo G.: *Computational Algorithms for Closed Queueing Networks*, North Holland , 1980.
- [BGJ92] Bolch G. , Gaebell M. , Jung. H.: *Entwicklung und Validierung der Schliessmethode zur Analyse offener Warteschlangennetze*, Operations Research Proceedings, Aachen, Springer, 1992.
- [Gaeb92] Gaebell M.: *Entwicklung und Validierung der Schliessmethode zur Analyse offener Warteschlangennetze*, Studienarbeit at IMMD IV University Erlangen, 1992.
- [Grei93] Greiner S.: *Leistungsbewertung des Betriebssystems UNIX mit Hilfe approximativer analytischer Methoden*, Studienarbeit at IMMD IV University Erlangen, 1993.
- [Grei95] Greiner S.: *An Analytical Model for the Operating System BS2000*, Diplomarbeit at IMMD IV University Erlangen, 1995.
- [Jung91] Jung H.: *Leistungsbewertung UNIX basierter Betriebssysteme für Multiprozessoren mit globalem Speicher*, Dissertation at IMMD IV University Erlangen, 1991.
- [Kauf84] Kaufmann J.S.: *Approximation Methods for Networks of Queues with Priorities*, Performance Evaluation Vol 4 , pp. 183 - 198 , 1984.
- [Kirs94] Kirschnik M.: *The Performance Evaluation and Prediction System for Queueing Networks PEPSY-QNS*, Technical Report TR-I4-18-94 of the Computer Science Department, IMMD IV, Operating Systems, University Erlangen, Germany, 1994
- [Kueh79] Kuehn P.J.: *Approximate Analysis of General Queueing Networks by Decomposition*, IEEE Transactions on Communications, Vol. 27, No1, pp. 113-126, Jan. 1979.
- [LZGS86] Lazowska E.D. , Zahorjan J. , Graham G.S. , Sevcik K.C.: *Quantitative System Performance - Computer System Analysis Using Queueing Network Models*, Prentice Hall , 1984.
- [Mari79] Marie R.: *An Approximate Analytical Method for General Queueing Networks* IEEE Transactions on Software Engineering, Vol.5, No.5, pp.530-538, Sept. 1979.
- [Munt73] Muntz R.R.: *Poisson Departure Process and Queueing Networks*, Proc. of the 7th Annual Princeton Conf. on Information Sciences and Systems , Princeton University pp 435 - 440 March 1973.
- [ReLa80] Reiser M. , Lavenberg S.S.: *Mean-Value-Analysis of Closed Multichain Queueing Networks*, Journal of the ACM , Vol 27 , No. 2 , pp 313 - 322 . April 1980.
- [SaCh81] Sauer C.H. , Chandy K.M.: *Computer System Performance Modelling*, Prentice Hall, 1981.
- [Sevc77] Sevcik K.C.: *Priority Scheduling Disciplines in Queueing Network Models of Computer Systems*, Proc. IFIP Congress , North Holland , pp 565 - 570 , 1977.