Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра ИКСС

Контрольная работа

по дисциплине: «Операционные системы»

Вариант 11

Выполнил: студент

группы *NN*

*ФИО*

Проверил:

*ФИО*

**Теория:**

Планирование процессов включает в себя решение следующих задач:

1. определение момента времени для смены выполняемого процесса;
2. выбор процесса на выполнение из очереди готовых процессов;
3. переключение контекстов "старого" и "нового" процессов.

Первые две задачи решаются программными средствами, а последняя в значительной степени на аппаратном уровне.

Существует множество различных алгоритмов планирования процессов, по-разному решающих вышеперечисленные задачи, преследующих различные цели и обеспечивающих различное качество мультипрограммирования. Среди этого множества алгоритмов рассмотрим подробнее две группы наиболее часто встречающихся алгоритмов: алгоритмы, основанные на квантовании, и алгоритмы, основанные на приоритетах.

В соответствии с алгоритмами, основанными на квантовании, смена активного процесса происходит, если:

1. процесс завершился и покинул систему,
2. произошла ошибка,
3. процесс перешел в состояние ОЖИДАНИЕ,
4. исчерпан квант процессорного времени, отведенный данному процессу.

Процесс, который исчерпал свой квант, переводится в состояние ГОТОВНОСТЬ и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый процесс из очереди готовых. Таким образом, ни один процесс не занимает процессор надолго, поэтому квантование широко используется в системах разделения времени. Граф состояний процесса, изображенный на рисунке 1, соответствует алгоритму планирования, основанному на квантовании.

Кванты, выделяемые процессам, могут быть одинаковыми для всех процессов или различными. Кванты, выделяемые одному процессу, могут быть фиксированной величины или изменяться в разные периоды жизни процесса. Процессы, которые не полностью использовали выделенный им квант (например, из-за ухода на выполнение операций ввода-вывода), могут получить или не получить компенсацию в виде привилегий при последующем обслуживании. По-разному может быть организована очередь готовых процессов: циклически, по правилу "первый пришел - первый обслужился" (FIFO) или по правилу "последний пришел - первый обслужился" (LIFO).

Другая группа алгоритмов использует понятие "приоритет" процесса. Приоритет — это число, характеризующее степень привилегированности процесса при использовании ресурсов вычислительной машины, в частности, процессорного времени: чем выше приоритет, тем выше привилегии.

Приоритет может выражаться целыми или дробными, положительным или отрицательным значением. Чем выше привилегии процесса, тем меньше времени он будет проводить в очередях. Приоритет может назначаться директивно администратором системы в зависимости от важности работы или внесенной платы, либо вычисляться самой ОС по определенным правилам, он может оставаться фиксированным на протяжении всей жизни процесса либо изменяться во времени в соответствии с некоторым законом. В последнем случае приоритеты называются динамическими.

Существует две разновидности приоритетных алгоритмов: алгоритмы, использующие относительные приоритеты, и алгоритмы, использующие абсолютные приоритеты.

В обоих случаях выбор процесса на выполнение из очереди готовых осуществляется одинаково: выбирается процесс, имеющий наивысший приоритет. По-разному решается проблема определения момента смены активного процесса. В системах с относительными приоритетами активный процесс выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние ОЖИДАНИЕ (или же произойдет ошибка, или процесс завершится). В системах с абсолютными приоритетами выполнение активного процесса прерывается еще при одном условии: если в очереди готовых процессов появился процесс, приоритет которого выше приоритета активного процесса. В этом случае прерванный процесс переходит в состояние готовности. На рисунке 2 показаны графы состояний процесса для алгоритмов с относительными (а) и абсолютными (б) приоритетами.

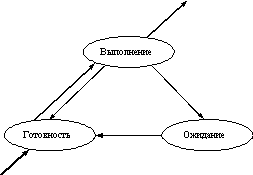


Рис.1 - Граф состояний процесса в многозадачной среде.

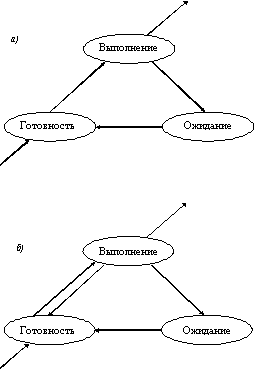


Рис.2 - Графы состояний процессов в системах

(а) с относительными приоритетами; (б)с абсолютными приоритетами

Во многих операционных системах алгоритмы планирования построены с использованием как квантования, так и приоритетов. Например, в основе планирования лежит квантование, но величина кванта и/или порядок выбора процесса из очереди готовых определяется приоритетами процессов.

Вытесняющие и невытесняющие алгоритмы планирования

Существует два основных типа процедур планирования процессов - вытесняющие (preemptive) и невытесняющие (non-preemptive).

Non-preemptive multitasking - невытесняющая многозадачность — это способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику операционной системы для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс.

Preemptive multitasking - вытесняющая многозадачность — это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

Понятия preemptive и non-preemptive иногда отождествляются с понятиями приоритетных и бесприоритетных дисциплин, что совершенно неверно, а также с понятиями абсолютных и относительных приоритетов, что неверно отчасти. Вытесняющая и невытесняющая многозадачность — это более широкие понятия, чем типы приоритетности. Приоритеты задач могут как использоваться, так и не использоваться и при вытесняющих, и при невытесняющих способах планирования. Так в случае использования приоритетов дисциплина относительных приоритетов может быть отнесена к классу систем с невытесняющей многозадачностью, а дисциплина абсолютных приоритетов - к классу систем с вытесняющей многозадачностью. А бесприоритетная дисциплина планирования, основанная на выделении равных квантов времени для всех задач, относится к вытесняющим алгоритмам.

Основным различием между preemptive и non-preemptive вариантами многозадачности является степень централизации механизма планирования задач. При вытесняющей многозадачности механизм планирования задач целиком сосредоточен в операционной системе, и программист пишет свое приложение, не заботясь о том, что оно будет выполняться параллельно с другими задачами. При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения активной задачи, запоминает ее контекст, выбирает из очереди готовых задач следующую и запускает ее на выполнение, загружая ее контекст.

При невытесняющей многозадачности механизм планирования распределен между системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама определяет момент завершения своей очередной итерации и передает управление ОС с помощью какого-либо системного вызова, а ОС формирует очереди задач и выбирает в соответствии с некоторым алгоритмом (например, с учетом приоритетов) следующую задачу на выполнение. Такой механизм создает проблемы как для пользователей, так и для разработчиков.

Для пользователей это означает, что управление системой теряется на произвольный период времени, который определяется приложением (а не пользователем). Если приложение тратит слишком много времени на выполнение какой-либо работы, например, на форматирование диска, пользователь не может переключиться с этой задачи на другую задачу, например, на текстовый редактор, в то время как форматирование продолжалось бы в фоновом режиме. Эта ситуация нежелательна, так как пользователи обычно не хотят долго ждать, когда машина завершит свою задачу.

Поэтому разработчики приложений для non-preemptive операционной среды, возлагая на себя функции планировщика, должны создавать приложения так, чтобы они выполняли свои задачи небольшими частями. Например, программа форматирования может отформатировать одну дорожку дискеты и вернуть управление системе. После выполнения других задач система возвратит управление программе форматирования, чтобы та отформатировала следующую дорожку. Подобный метод разделения времени между задачами работает, но он существенно затрудняет разработку программ и предъявляет повышенные требования к квалификации программиста. Программист должен обеспечить "дружественное" отношение своей программы к другим выполняемым одновременно с ней программам, достаточно часто отдавая им управление. Крайним проявлением "недружественности" приложения является его зависание, которое приводит к общему краху системы. В системах с вытесняющей многозадачностью такие ситуации, как правило, исключены, так как центральный планирующий механизм снимет зависшую задачу с выполнения.

Однако распределение функций планировщика между системой и приложениями не всегда является недостатком, а при определенных условиях может быть и преимуществом, потому что дает возможность разработчику приложений самому проектировать алгоритм планирования, наиболее подходящий для данного фиксированного набора задач. Так как разработчик сам определяет в программе момент времени отдачи управления, то при этом исключаются нерациональные прерывания программ в "неудобные" для них моменты времени. Кроме того, легко разрешаются проблемы совместного использования данных: задача во время каждой итерации использует их монопольно и уверена, что на протяжении этого периода никто другой не изменит эти данные. Существенным преимуществом non-preemptive систем является более высокая скорость переключения с задачи на задачу.

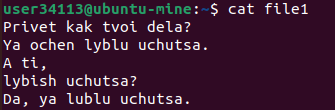
Примером эффективного использования невытесняющей многозадачности является файл-сервер NetWare, в котором, в значительной степени благодаря этому, достигнута высокая скорость выполнения файловых операций. Менее удачным оказалось использование невытесняющей многозадачности в операционной среде Windows 3.х.

Однако почти во всех современных операционных системах, ориентированных на высокопроизводительное выполнение приложений (UNIX, Windows NT, OS/2, VAX/VMS), реализована вытесняющая многозадачность. В последнее время дошла очередь и до ОС класса настольных систем, например, OS/2 Warp и Windows 95. Возможно, в связи с этим вытесняющую многозадачность часто называют истинной многозадачностью.

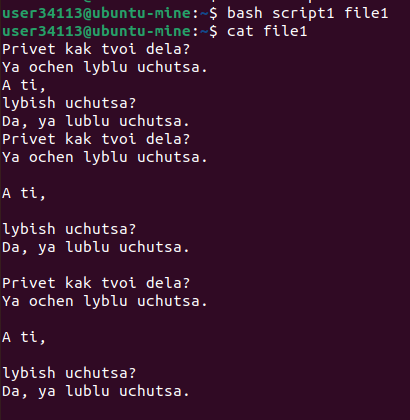
**Источники**

1. *Dfe «Управление процессами»* [электронный ресурс] // <https://dfe.petrsu.ru/linux/olifer/glava_6.htm>
2. *CoderLessons.com «Алгоритмы планирования операционной системы – CoderLessons.com»* [электронный ресурс] // <https://coderlessons.com/tutorials/akademicheskii/izuchite-operatsionnuiu-sistemu/algoritmy-planirovaniia-operatsionnoi-sistemy>

Задание 1.

  
Изображение выглядит как текст

Автоматически созданное описание



count\_check = 0 # переменная для подсчёта количества строчек заканчивающихся на точку или запятую  
regex=’[.,]$’ # регулярное выражение для точки и запятой в конце строки  
while read line # цикл по строчкам введённого файла

do

if [[ $line =~ $regex ]] # если строчка совпадает с регулярным выражением

then # то

echo “$line” >> buff\_file.txt # вывод строчки в буфер-файл

echo “” >> buff\_file.txt # добавление после строчки пустой стройки в буфер-файл

count=`expr $count + 1` # увеличение счётчика количества строчек заканчивающихся на точку или запятую на 1

else # иначе

echo “$line” >> buff\_file.txt # вывод строчки в буфер-файл

fi

done < “$1” # передача в цикл содержимого файла переданного первым аргументов командной строки

if [ $count -eq 0 ] # если количество нужных строчек равно 0

then # то

echo “In file not str end on , and/or .” # вывод информации о том, что в файле нет таких строчек вовсе

rm buff\_file.txt # удаление файла буфера

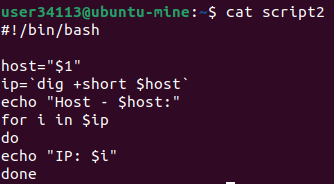
else # иначе

cp buff\_file.txt “$1” # замена копированием содержимого изначального файла на файл буфер

rm buff\_file.txt # удаление файла буфера

fi

Задание 2.

  
  
Изображение выглядит как текст

Автоматически созданное описание

host=”$1” # в переменную host записывается введённый первым параметром значение

ip=`dig +short $host` # команда dig выводит информацию о хосте, ключ short выводит только ip-шники хоста в переменной хост

echo “Host - $host:”

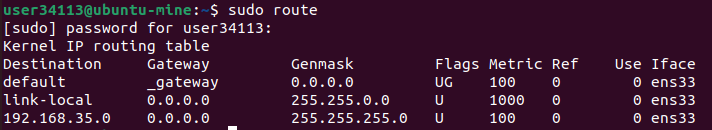
for i in $ip # цикл по айпи в переменной ip

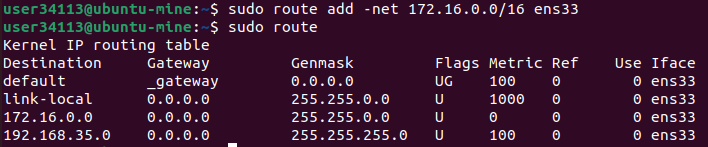
do

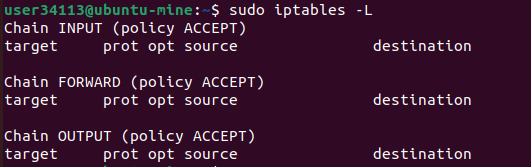
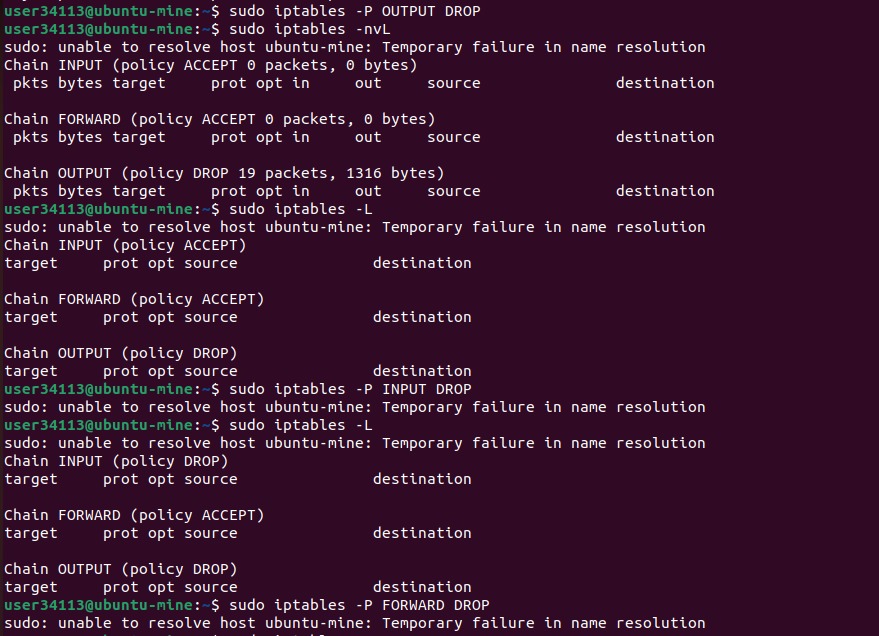
echo “IP: $i”

done

Задание 3.

* 1. route add -net 172.16.0.0/16 ens33



* 1.   
       
     Изображение выглядит как текст

     Автоматически созданное описание  
     iptables – P OUTPUT DROP # все соединения цепочки OUTPUT блокировать

iptables -P INPUT DROP # все соединения цепочки INPUT блокировать  
iptables -P FORWARD DROP # все соединения цепочки FORWARD блокировать

iptables -A OUTPUT -d 172.168.0.0/16 -p udp –dport 67:68 –sport 67:68 -j ACCEPT # исходящие соединения dhcp (67 68 port) с хостов, к которым настроена маршрутизация в 1 пункте 3 задания разрешить.

iptables -A INPUT -s 172.168.0.0/16 -p udp –dport 67:68 –sport 67:68 -j ACCEPT # входящие соединения dhcp (67 68 port) с хостов, к которым настроена маршрутизация в 1 пункте 3 задания разрешить.