Report

Operating Systems and Networks

—

Likhith Asapu
2020114015
UG3 CLD

Sudha Tanay.D
2020115010
UG3 CHD

Question Statement

1) Include a well-written description of how you put the other four standards into practice.

2) The report must contain an explanation about the implementation of the various implemented scheduling algorithms.

3) Include the performance comparison between the default and 3 implemented policies in the README by showing the average waiting and running times for processes

Report

1) Implementing System calls (Strace & Sigalarm-Sigreturn)

In order to implement strace, several key steps were done:

a) A new user program entry '**$U/_strace**' was added to the makefile.

b) A variable to store the tracemask called 'mask' was added to **kernel/proc.h**. In order to save the value of the mask across parent and child processes, the fork() function in **kernel/proc.c** was modified to copy the mask value from the parent to child process.
c) The function sys_trace() was added to **kernel/sysproc.c,** which moves the mask value from user space to kernel space by saving it as a variable called 'mask.'
d) The user program **user/strace.c** was created, which ensures that the input is in the correct format and traces the system calls called by the command provided in the prompt.
e) The syscall() function in **kernel/syscall.c** was modified to process the mask in a bitwise fashion and trace the corresponding syscalls defined in the same file itself under syscallnames[] array created to keep track of the list of syscalls. If the syscall does not exist, an error is thrown, and a syscall that exists has its corresponding mask value checked in order to trace it using bitwise AND after the right-shift operation.

In order to implement sigalarm and sigreturn, these steps were done:

a) A variable called 'handler' and 'handlerpermission' was added to **kernel/proc.h.** 'handlerpermission' variable acts as a lock and prevents two different CPU cores from triggering the signal alarm at once. Other variables, such as 'alarm_tf' (to keep the current state of registers in a trap frame when alarm handler is called) and 'alarm_on' (to set the status of alarm if sigalarm syscall is called), and variables for tick count and tick interval were added in the form of 'cur_ticks' and 'ticks.'
b) The function sys_sigalarm() was added to **kernel/sysproc.c,** which moves the ticks, handler, and alarm_on value from user space to kernel space by saving it in its corresponding variables
c) The usertrap() function in **kernel/trap.c** was modified to trigger the alarm periodically.
d) The function sys_sigreturn() was added to **kernel/sysproc.c,** which restores the process state to before the alarm handler was called.

2) Implementing Scheduling algorithms

FCFS:

a) The scheduling algorithm first initializes a dummy first process in the form of 'firstproc' and goes through the list of processes. The process with the earlier time creation (which is calculated by tick value) will be assigned the first process status.
b) Once the first process is determined, its state is set to RUNNABLE and the process executes without preemption. The non-preemtpion was done by modifying and adding a conditional entry to the yield() function in **kernel/trap.c**

c) Once it has been executed, a new 'firstproc' is selected from the remaining list of scheduled processes.
d) Its implementation is in **kernel/proc.c** in the scheduler() function

LOTTERY:

a) The settickets syscall implemented changes the number of tickets (based on arguments passed) for a particular process
b) The scheduling algorithm first calculates the cumulative total of tickets held by all runnable processes.
c) A random value is selected between 0 and the cumulative ticket total (stored in totalticketval) and stored in variable ticketval.
d) A process is selected if the randomly generated number falls within the bounds of tickets owned by a process. It is executed and preempted if necessary.
e) Its implementation is in **kernel/proc.c** in the scheduler() function

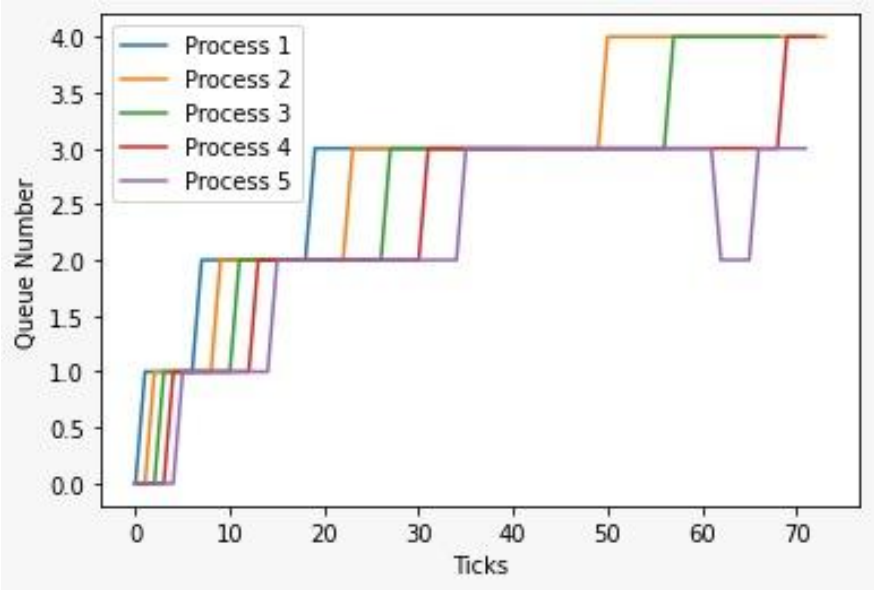
MLFQ:

a) Rather than creating 5 separate queues, each process has a variable that signifies which queue it is in. Therefore, when a process moves between queues, the variable simply gets updated depending on promotion or demotion.
b) The processes initially enter queue 0, which is the queue with the lowest time quanta, and the processes are scheduled according to preemptive FCFS. This FCFS also applies to queues 1,2,3. The last queue (queue 4) is RoundRobin.
c) If a process enters a queue that is higher than the queue value of the currently running process, the current process is preempted.
d) Processes that compute for long are demoted(applies to queues 0,1,2,3), and processes that wait for long are promoted.
e) Its implementation is in **kernel/proc.c** in the scheduler() function

PBS:

a) Using setpriority system call implemented, the default priority of processes can be set to 60
b) Dynamic priority is calculated from static priority using niceness value, which is initially set to 5 when the process is created but would change as the process sleeps.
c) When processes with the same dynamic priority exist, the tie is broken by first comparing the number of times the process is scheduled and then the creation time of the process.
d) Its implementation is in **kernel/proc.c** in the scheduler() function

MLFQ Scheduling Analysis Graph



Aging time: 30 ticks