**Homework 2 (7.5%)**

CSE 4600 (Section ##) – Operating Systems – Fall 2021

Submitted to

Department of Computer Science and Engineering

California State University, San Bernardino, California

by

Ulises Perez Priego (007497809)

Date of issue: October 18, 2021

Due date: October 24, 2021, 11:59 pm

*No copy and paste from other colleagues with the same answers and description (particularly in part 3) is allowed. It is required that you carry out the exercises by yourself (with the possibility for collaborating with other colleagues) and provide descriptions (with screenshots wherever necessary) based on your own experience. Copied material from other colleagues will be considered as cheating and dealt with seriously through University academic integrity policies.*

Email: 007497809@coyote.csusb.edu

**Note:** Assume all the times in the exercises below are in milliseconds. Please report your results (average waiting and turnaround times) with precision of up to two significant digits.

1. **Preemptive shortest remaining time first (SRTF) and Round Robin scheduling algorithms – 2%**
2. Consider the processes with given arrival and burst times as below

|  |  |  |
| --- | --- | --- |
| Process ID | Arrival Time | Burst Time |
| P1 | 0 | 8 |
| P2 | 0 | 9 |
| P3 | 2 | 12 |
| P4 | 4 | 3 |

Assume the OS is using the shortest remaining time first (SRTF) algorithm. Calculate the waiting and turnaround times for each process. Provide the Gantt chart below to show the execution of the processes and fill in the columns in the table below for the waiting time and turnaround time

Gantt Chart: Insert your Gantt chart below

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P4 | P1 | P2 | P3 |

0 4 7 11 20 32

Turnaround time and waiting time

|  |  |  |
| --- | --- | --- |
| Process ID | Waiting Time | Turnaround Time |
| P1 | 3 | 11 |
| P2 | 11 | 20 |
| P3 | 18 | 30 |
| P4 | 0 | 3 |

Average waiting time = (3+11+18+0)/4 = 8ms

Average turnaround time = (11+20+30+3)/4 = 16ms

1. Consider the processes with given burst times as below and same arrival time as 0ms

|  |  |
| --- | --- |
| Process ID | Burst Time |
| P1 | 8 |
| P2 | 5 |
| P3 | 13 |

Assume the OS is using Round Robin scheduling algorithm with Time quantum value of 4ms. Calculate the waiting and turnaround times for each process. Provide the Gantt chart below to show the execution of the processes and fill in the columns in the table below for the waiting time and turnaround time

Gantt Chart: Insert your Gantt chart below

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P1 | P2 | P3 | P3 | P3 |

0 4 8 12 16 17 21 25 26

Turnaround time and waiting time

|  |  |  |
| --- | --- | --- |
| Process ID | Waiting Time | Turnaround Time |
| P1 | 8 | 16 |
| P2 | 12 | 17 |
| P3 | 13 | 26 |

Average waiting time = (8+12+13)/3 = 11ms

Average turnaround time = (16+17+26)/3 = 19.67ms

1. **Priority Scheduling algorithm (non-preemptive) – 2%**

Consider the processes with given burst times and priorities as below

|  |  |  |
| --- | --- | --- |
| Process ID | Burst time | Priority |
| P1 | 12 | 2 |
| P2 | 14 | 0 |
| P3 | 6 | 1 |
| P4 | 7 | 3 |

Assume the OS is using non-preemptive priority scheduling algorithm. Calculate the waiting and turnaround times for each process. Provide the Gantt chart below to show the execution of the processes and fill in the columns in the table below for the waiting time and turnaround time

Gantt Chart: Insert your Gantt chart below

|  |  |  |  |
| --- | --- | --- | --- |
| P2 | P3 | P1 | P4 |

0 14 20 32 39

Turnaround time and waiting time

|  |  |  |
| --- | --- | --- |
| Process ID | Waiting Time | Turnaround Time |
| P1 | 20 | 32 |
| P2 | 0 | 14 |
| P3 | 14 | 20 |
| P4 | 32 | 39 |

Average waiting time = 16.5

Average turnaround time = 26.25

1. **xv6 -Implementation of ps, nice system calls and priority scheduling – 3.5%**

Article used to prepare this section: [Link to the article](https://medium.com/@harshalshree03/xv6-implementing-ps-nice-system-calls-and-priority-scheduling-b12fa10494e4)

Youtube link for xv6 scheduling: [YouTube link here](https://www.youtube.com/watch?v=hIXRrv-cBA4&t=25s)

The assumption by now is that you are familiar with the xv6 operating system and its basics.

**ps and nice system calls walkthrough**

The ps (i.e., process status) command is used to provide information about the currently running processes, including their process identification numbers (PIDs).

The nice system call is used to change the priority of a given process.

1. System call interface maintains the table of all the system call and associates each with a number. Update this table. In **syscall.h**, add the following two system calls. Here, the numbers 22 and 23 are given to the system calls. The cps is for the ps system call and chpr (change priority) is for the nice system call.

#define SYS\_cps 22

#define SYS\_chpr 23

1. In the PCB of the process, add a new attribute ‘priority’. The PCB of the process is stored in **proc.h** file.

**In the struct proc in the proc.h file, add a new attribute ‘priority’ of int data type.**

1. Include the declaration of these functions in **defs.h** and **users.h** files.

|  |  |  |
| --- | --- | --- |
| //Add this below the //proc.c section in defs.h | | |
|  | | |  |
|  | // proc.c | | |
|  | int cps(void); | | |
|  | int chpr(int pid, int priority); | | |
| //Add this below the system calls section in users.h | | | | |
|  | | int cps(void); | | | |
|  | | int chpr(int pid, int priority); | | | |

1. Include the definition of the cps and chpr functions in ***proc.c***

//Add this in the end of the proc.c file

|  |
| --- |
| int |
|  | cps() |
|  | { |
|  | struct proc \*p; |
|  | //Enables interrupts on this processor. |
|  | sti(); |
|  |  |
|  | //Loop over process table looking for process with pid. |
|  | acquire(&ptable.lock); |
|  | cprintf("name \t pid \t state \t priority \n"); |
|  | for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){ |
|  | if(p->state == SLEEPING) |
|  | cprintf("%s \t %d \t SLEEPING \t %d \n ", p->name,p->pid,p->priority); |
|  | else if(p->state == RUNNING) |
|  | cprintf("%s \t %d \t RUNNING \t %d \n ", p->name,p->pid,p->priority); |
|  | else if(p->state == RUNNABLE) |
|  | cprintf("%s \t %d \t RUNNABLE \t %d \n ", p->name,p->pid,p->priority); |
|  | } |
|  | release(&ptable.lock); |
|  | return 22; |
|  | } |
|  |  |
|  | int |
|  | chpr(int pid, int priority) |
|  | { |
|  | struct proc \*p; |
|  | acquire(&ptable.lock); |
|  | for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){ |
|  | if(p->pid == pid){ |
|  | p->priority = priority; |
|  | break; |
|  | } |
|  | } |
|  | release(&ptable.lock); |
|  | return pid; |
|  | } |

1. In ***sysproc.c***, define a function in which cps and chpr functions will be called.

//Add this in the end of the sysproc.c file

|  |
| --- |
| int |
|  | sys\_cps(void) |
|  | { |
|  | return cps(); |
|  | } |
|  |  |
|  | int |
|  | sys\_chpr(void) |
|  | { |
|  | int pid, pr; |
|  | if(argint(0, &pid) < 0) |
|  | return -1; |
|  | if(argint(1, &pr) < 0) |
|  | return -1; |
|  |  |
|  | return chpr(pid, pr); |
|  | } |

1. Make minor changes in the usys.S file. The ‘.S’ extension indicates that this file has assembly level code and this file interacts with the hardware of the system.

//Add this in the end of the usys.S file

SYSCALL(cps)

SYSCALL(chpr)

1. Open sysproc.c file and add the two system calls.

//Add this where the other system calls are defined in syscall.c

extern int sys\_cps(void);

extern int sys\_chpr(void);

|  |
| --- |
| /\* |
|  | . |
|  | . |
|  | . |
|  | \*/ |
|  |  |
|  | //Add this inside static int (\*syscalls[])(void) |
|  | [SYS\_cps] sys\_cps, |
|  | [SYS\_chpr] sys\_chpr, |

1. Create a ***ps.c*** and ***nice.c*** file in which the cps and chpr functions will be called respectively.

|  |
| --- |
| #include "types.h" |
|  | #include "stat.h" |
|  | #include "user.h" |
|  | #include "fcntl.h" |
|  |  |
|  | int main(void){ |
|  | cps(); |
|  | exit(); |
|  | } |

|  |
| --- |
| #include "types.h" |
|  | #include "stat.h" |
|  | #include "user.h" |
|  | #include "fcntl.h" |
|  |  |
|  | int |
|  | main(int argc, char \*argv[]) |
|  | { |
|  | int priority, pid; |
|  | if(argc < 3){ |
|  | printf(2,"Usage: nice pid priority\n"); |
|  | exit(); |
|  | } |
|  | pid = atoi(argv[1]); |
|  | priority = atoi(argv[2]); |
|  | if (priority < 0 || priority > 20){ |
|  | printf(2,"Invalid priority (0-20)!\n"); |
|  | exit(); |
|  | } |
|  | chpr(pid, priority); |
|  | exit(); |
|  | } |

1. The work on the system calls is now done. Now work on the process priority assignment. For this, define the default priority of a process in the allocproc function in the **proc.c** file.

**As per the class discussions, the higher the number indicates lower priority of the process.**

//Add this under the "found:" part of the allocproc function in proc.c

|  |
| --- |
| found: |
|  | p->state = EMBRYO; |
|  | p->pid = nextpid++; |
|  | p->priority = 10; //Default Priority of a process is set to be 10 |
|  |  |

1. The child process is now expected to have higher priority than the parent process. The priority of child process needs to be changed when it is created. For this, make the changes in ***exec.c*** file.

|  |  |
| --- | --- |
|  | /\* Add this above the "bad:" part in the exec.c file where all other child process attributes are mentioned \*/ |

curproc->priority = 2; //Giving child process default priority of 2

1. Create a c program which creates a number of child process as mentioned by the user and consumes CPU time for testing the system calls and scheduling. Create a new file ***dummy\_func.c(dummy program)*** and write the following code:

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fcntl.h"

int main(int argc, char \*argv[]) {

int pid;

int k, n;

int x, z;

if(argc < 2)

n = 1; //Default

else

n = atoi(argv[1]);

if (n < 0 ||n > 20)

n = 2;

x = 0;

pid = 0;

for ( k = 0; k < n; k++ ) {

pid = fork ();

if ( pid < 0 ) {

printf(1, "%d failed in fork!\n", getpid());

} else if (pid > 0) {

// parent

printf(1, "Parent %d creating child %d\n",getpid(), pid);

wait();

}

else{

printf(1,"Child %d created\n",getpid());

for(z = 0; z < 4000000000; z+=1)

x = x + 3.14\*89.64; //Useless calculation to consume CPU Time

break;

}

}

exit();

}

1. Make the appropriate changes in the ***Makefile***. In Makefile, under ‘UPROGS’, add the following:

\_ps\   
\_dummy\_func\   
\_nice\

1. In the EXTRAS section of the Makefile, add**nice.c**, **dummy\_func.c** and **ps.c**

**Priority based round robin scheduling walkthrough**

Priority based Round-Robin CPU Scheduling algorithm is based on the integration of round-robin and priority scheduling algorithm. It retains the advantage of round robin in reducing starvation and integrates the advantage of priority scheduling.

1. For implementing this, make the required changes in scheduler function in **proc.c** file.

//Replace the scheduler function with the one below for priority round robin scheduling

void

scheduler(void)

{

struct proc \*p, \*p1;

struct cpu \*c = mycpu();

c->proc = 0;

for(;;){

// Enable interrupts on this processor.

sti();

struct proc \*highP;

// Loop over process table looking for process to run.

acquire(&ptable.lock);

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if(p->state != RUNNABLE)

continue;

// Switch to chosen process. It is the process's job

// to release ptable.lock and then reacquire it

// before jumping back to us.

highP = p;

//choose one with highest priority

for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){

if(p1->state != RUNNABLE)

continue;

if(highP->priority > p1->priority) //larger value, lower priority

highP = p1;

}

p = highP;

c->proc = p;

switchuvm(p);

p->state = RUNNING;

swtch(&(c->scheduler), p->context);

switchkvm();

// Process is done running for now.

// It should have changed its p->state before coming back.

c->proc = 0;

}

release(&ptable.lock);

}

}

At this step, you have implemented the system calls and changed the scheduling policy in xv6. Now, let us try it out.

**Work to do:**

1. No more than 1 page: Summarize the steps you took above in a paragraph to describe the process to change the priority of a process in xv6 operating system.
2. Call the dummy\_func functions twice simultaneously which creates a process scenario. Screenshot your command prompt and copy it here.
3. Initially, the “dummy\_func” function with pid 9 should be in “RUNNABLE” state with the default priority 10. Show a screenshot of this step.
4. Using the nice system call, the priority of the process with pid 9 gets changed to 1 which makes the that process a higher priority.
5. Call the ps command again and report the states of the process with pid 9 state and its previous running process which should now be “RUNNING” state.