클립아트, 만화 영화이(가) 표시된 사진

자동 생성된 설명

COSE341(01)

Operating system

Project #2 Report

Your name : Yeyoung Park

Student # : 2020160027

YOUR DEPARTMENT : Mathematics

Submission Date : 2024.11.14

Number of free days used for this project: 1 day

# Development Environment

* 1. Host OS : Windows 10
  2. VirtualBox : version 7.1.2
  3. Ubuntu : 18.04.2 LTS
  4. Linux Kernel : 4.20.11.sslab

# Explanation of CPU scheduling and policies [20 pts]

1. Describe the need for CPU scheduling (4 pts)

The need for CPU scheduling arises due to many reasons. First, it can make efficient use of CPU resources. Only one process can use the CPU. Scheduling ensures the CPU is always assigned to a process, reducing idle time. Also, by efficiently managing process execution order, scheduling increases the number of processes completed per unit time, so, throughput enhanced. Second, it enables the OS to switch between processes rapidly, giving the illusion that multiple processes are running simultaneously. Also, it ensures that user applications respond promptly. Third, it prioritize processes. Some processes are time-sensitive, so, scheduling can prioritize these processes to meet specific timing requirements. By assigning priorities, the system can provide different levels of service to different applications. Fourth, it can help reduce power usage by managing CPU workload and enabling low-power state. In summary, it is necessary to manage the limited CPU resources efficiently in a multitasking environment. It ensures that all running processes get a fair share of the CPU. Also, it optimizes system performance and enhances the user experience.

1. Describe advantages and disadvantages of First-Come, First-Served [FCFS] scheduling (8 pts)

It is easy to implement and does not need to perform complex calculations or maintain data structures, reducing the CPU time spent on scheduling tasks. However, it has poor average waiting times. If short processes are behind long processes, it can stuck. (known as Convoy effect) So, critical tasks can be delayed. Also basically, it is non-preemptive scheduling algorithms, may cause starvation of processes.

1. Describe advantages and disadvantages of Shortest-Job-First [SJF] scheduling (8pts)

It can reduce average waiting times and turnaround times than FCFS. However, in real world, we don’t know the exact CPU burst time of processes in many cases, can make hard to implement SJF effectively. Also, same as FCFS, it is non-preemptive scheduling algorithms basically, cause starvation of processes. Also, it has more scheduling overheads than FCFS.

# Implementation [40 pts]

# A, B, D, E: Please attach only added or edited part of code in text.

# C: Describe the workflow focusing on the decision made by the scheduling policy for the implemented system call.

### (linux)/arch/x86/entry/syscalls/syscall\_64.tbl (4 pts)

# yeyoung --- start

335 common os2024\_push \_\_x64\_sys\_os2024\_push

336 common os2024\_pop \_\_x64\_sys\_os2024\_pop

339 common os2024\_ku\_cpu \_\_x64\_sys\_os2024\_ku\_cpu

# yeyoung --- end

### (linux)/include/linux/syscalls.h (4 pts)

/\* yeyoung --- start \*/

asmlinkage int sys\_os2024\_push(int a);

asmlinkage int sys\_os2024\_pop(void);

asmlinkage int sys\_os2024\_ku\_cpu(char \*name, int jobTime);

/\* yeyoung --- end \*/

### (linux)/kernel/sslab\_ku\_cpu.c (20 pts)

When a process invokes the system call ku\_cpu with a job time (jobTime > 0), the system first checks whether the CPU is currently idle (now == IDLE) or occupied by another process. If the CPU is idle (no process is currently using the CPU), the system grants the calling process immediate access to the CPU. The now variable is set to the current process's PID, indicating it has claimed the CPU, and the process begins its work without entering the waiting queue. If the CPU is already occupied by another process (now != IDLE and now != current->pid), the calling process cannot immediately access the CPU. The system then checks if this competing process is already in the waiting queue using the is\_in\_queue() function. If the process is not already in the queue, it is added to the waiting queue along with its job time. The system then denies this process’s request to use the CPU at this moment, enforcing the waiting policy, and it logs that the process has been added to the queue. When a process’s job completes (jobTime == 0), it signals that it has finished using the CPU. The handler Marks the CPU as idle by resetting now to IDLE and frees any allocated memory related to this process, as it no longer needs CPU time. When the CPU becomes idle after a process completes, the system checks the waiting queue for the next process. Using the Shortest Job First (SJF) policy, the system selects the process in the waiting queue with the smallest jobTime. This is accomplished by iterating through the queue and identifying the process with the shortest job time. Using the First-Come First-Served (FCFS) policy, the system selects the front process in the waiting queue. Once identified, this process is removed from the queue and assigned to the CPU. This process then begins its execution. The handler continues this cycle, repeatedly assigning the CPU to the process with the shortest job time each time in the case of SJF, in order of first come time in the case of FCFS when the CPU becomes available.

### (linux)/kernel/Makefile (4 pts)

obj-y = fork.o exec\_domain.o panic.o \

cpu.o exit.o softirq.o resource.o \

sysctl.o sysctl\_binary.o capability.o ptrace.o user.o \

signal.o sys.o umh.o workqueue.o pid.o task\_work.o \

extable.o params.o \

kthread.o sys\_ni.o nsproxy.o \

notifier.o ksysfs.o cred.o reboot.o \

async.o range.o smpboot.o ucount.o sslab\_my\_stack.o sslab\_ku\_cpu.o

### /project2/kucpu\_cpu.c (8 pts)

#include <unistd.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define KU\_CPU 339 // Define the syscall number

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

int jobTime;

int delayTime;

char name[4];

int wait = 0;

int responseTime = -1; // Initialize response time to -1 to check the first successful access

(…)

// Continue requesting the system call as long as jobTime remains

while (jobTime > 0) {

int result = syscall(KU\_CPU, name, jobTime);

if (result == 0) {

// The request was accepted

jobTime--; // decrease job time on each successful access

// If this is the first successful request, set response time

if (responseTime == -1) {

responseTime = wait; // Response time is the wait count at the first acceptance

}

} else {

// The request was rejected, increase wait time

wait++;

}

usleep(100000); // delay 0.1 second

}

// Signal the completion of the job to ku\_cpu

syscall(KU\_CPU, name, 0);

printf("\nProcess %s: Finish! My response time is %.1fs and My total wait time is %.1fs.\n",

name, responseTime \* 0.1, wait \* 0.1);

return 0;

}

# Experiment results & Analysis [10 pts]

1. Draw a First-Come, First-Served [FCFS] scheduling line graph (2pts)

라인, 그래프, 번호, 도표이(가) 표시된 사진

자동 생성된 설명

1. Draw a Shortest-Job-First [SJF] scheduling line graph (2pts)

라인, 그래프, 번호, 텍스트이(가) 표시된 사진

자동 생성된 설명

1. Explain each scheduling method and compare of FCFS and SJF scheduling in terms of average wait time and response time (6 pts)

FCFS means First-come First-served. It is non-preemptive. It means that process change occurs only when a process give voluntary yield to another process. So, if a process being executed, it occupies CPU till its end. Since process A requested first, process B second, process C third, and process D last, process A occupies CPU first and executed till its end. After process A terminates, process B occupies CPU and executed till its end. After that, process C occupies CPU and executed till its end. Lastly, process D occupies CPU and executed till its end. So, wait time and response time are same in this case. Process A waits 0(s), process B waits 6.1(s), process C waits 10.2(s), process D waits 12.3(s). So the average wait time is (0 + 6.1 + 10.2 + 12.3) / 4 = 7.15(s).

SJF means Shortest Job First. In this method, shorted job in the waiting queue executed first. Also, it is non-preemptive. In 0(s), only process A requires, so, process A occupies CPU first and executed till its end. Remind that I implemented SJF, not SRTF (SJF with preemption). So, wait time and response time is same. After process A terminated, process B, C, D in the queue, but process D has shorter CPU burst time than B and C, so process D executed first. After process D terminated, process C has shorter CPU burst time than B, so, process C executed and terminated. After that, process B executed and terminated. Process A waits 0(s), process D waits 4(s), process C waits 6(s), process B waits 10.1(s). So the average wait time is (0 + 4 + 6 + 10.1) / 4 = 5.025(s).

We can conclude that SJF scheduling is better than FCFS scheduling in terms of average waiting time. Also, we know that this is not ideal situation, so, one process terminate time and another process begin to execute time is not same. Time gap always exists between them.

# Advanced Question [10 pts]

# Q. We execute three processes as shown in Table 1. There are no scheduling and context switching delays. Also, the unit of time is second (s). If I want my OS to terminate process B before process C, how should I set the time slice value T for the round robin scheduling? Find the smallest T value.

# 텍스트, 스크린샷, 라인이(가) 표시된 사진 자동 생성된 설명

# Ans.

# Note that in Round Robin scheduling, each process gets a time slice T (which we also called “Quantum”) of CPU time and is cycled through FCFS manner. In this moment, priorities are not considered in Round Robin scheduling. Also, if a process’s remaining CPU burst time is less than the time slice, it will finish execution in that turn.

# Let the time slice T = 11(s). Then, the process A will be executed and terminated in the time interval between 0(s) to 6(s) (by FCFS manner). After that, the process B will be executed in the time interval between 6(s) to 17(s). The process B has remaining 1(s) of CPU-burst time at that moment. After that, the process C will be executed and terminated in the time interval between 17(s) to 20(s). After that, the process B will be executed and terminated in the time interval 20(s) to 21(s). So, this time slice is not sufficient.

# Let the time slice T = 12(s). Then, the process A will be executed and terminated in the time interval between 0(s) to 6(s). After that, the process B will be executed and terminated in the time interval between 6(s) to 18(s). After that, the process C will be executed and terminated in the time interval between 18(s) and 21(s). So, if we set quantum T larger than or equal to 12(s), then the process B will be terminated before process C. Therefore, the smallest T value is 12(s).