Linux 운영체제 및 응용

Spring 2019 GITF315

강사: Dr. Youngjae Kim

중간고사 (Take-home) - 2019.04.26 (금요일)

- **제출기한: 2019.04.26 금요일 자정 (24:00)**
- **제출방법: 답안지 작성 후 스캔하여 이메일로 제출 (Email: youkim@sogang.ac.kr)**
  - **답안지 원본은 다음 수업시간에 제출**
- **답안지 작성시 유의사항**
  - **과목명, 학번, 전공, 성명 반드시 기재하여 작성**
  - **Take-home 시험 시간, 종료시간 표시 (시험 본 시간 표시)**
    - **Take-home 시험에 응한 시간은 점수에 반영하지 않음 (참고용)**
- **문항수: 8문항 (총점: 115점)**

1. True or False (14 points – 2 points each) (각 문제 답에 대한 이유를 설명하세요.)

(1) All I/O instructions are privileged instructions that can only be executed in kernel mode.

(2) In a Hosted VM (Virtual Machine), the virtual-machine monitor (VMM) runs on top of the guest operating system.

(3) Inter-process communication (IPC) refers to mechanisms that allow communications between multiple processes on the same machine.

(4) The degree of multiprogramming is the maximum number of processes that can be present in the ready queue. It is decided and controlled by the CPU scheduler.

(5) If threads are implemented using user-level thread library (Many-to-One mapping), the program cannot gain parallelism by running threads on multiple processors.

(6) FCFS scheduling may cause convoy effect, where processes are waiting in the ready queue for one CPU intensive process with a large burst time.

(7) A preemptive SJF scheduling may cause starvation, whereas starvation does not occur in non-preemptive SJF scheduling.

2. Terms (14 points – 2 points each)

(1) [   ] is a data structure in the operating system containing information needed to manage the scheduling of a particular process. It contains pieces of information associated with a specific process such as process state, CPU registers, priority, and memory locations.

(2) [ ] is a software-generated interrupt caused either by an error or by a specific request from a user program that an operating-system service be performed.

(3) [ ] is a computer program that is the core of a computer's operating system, with complete control over everything in the system. On most systems, it is one of the first programs loaded on start-up, after the bootloader. It is the also the one program running at all times on the computer.

(4) [ ] is a type of operating system structure where all non-essential components such as device drivers, file system, and application IPC are removed from the kernel and implemented as system and user-level programs. On the other hand, a monolithic kernel is another type of OS structure, where all those components are working in kernel space.

(5) [ ] is a task required when a CPU switches running processes. It involves storing the state of the old process (or thread), and loading the saved state of the new process.

(6) [ ] is a communication channel used for inter-process communication. It is an unnamed one-way communication channel that is typically used between a parent process and a child.

(7) [ ] is a process scheduling algorithm where there are multiple queues with different priorities, and a process may move between the queues.

3. Brief Answers (21 points – 3 points each)

(1) What are the differences between a process and a thread? Explain in perspectives of memory sharing, switching, communication, etc.

(2) In inter-process communication using message passing, there are two types of send/receive operations: blocking send/receive and non-blocking send/receive. What is the difference between the two approaches? Explain how the operations will behave when called.

(3) What is a thread pool? How does using thread pools help save performance costs of applications?

(4) What is a zombie process and an orphan process? Explain how they are created and how they are handled.

(5) As a process executes, it changes states. Explain what are ready state, running state, and waiting state. Also, describe when a process moves between these three states.

(6) When a process is created, memory is allocated for the process. Describe each section of the memory allocated to the process, and what goes into each section.

(7) We often use library functions such as fopen, fread, fwrite, and fclose instead of calling system call functions such as open, read, write, and close. What is the difference between calling a library function and calling a system call function?

4. Processes (16 points – (1) 5, (2) 5, (3) 6)

(1) Let us suppose we write a function like the one below.

```
void MyFork(int num_child) {
    int i;
    for(i = 0; i < num_child; i++) {
       pid = fork();
       if(pid == 0) printf("my pid is: %d\n", getpid());
                                                  //getpid() returns current pid
    }
```

We execute a program, which is given the process id 2000. After that, let us assume that pid is assigned in incremental order as new processes are created. Also, we assume that there is no other process. If we call **MyFork(10)** inside the program, what is the maximum pid that is printed on the display? Explain why.

(2) If we run the code below, what is printed on the display? Write your answer, and also explain why. (The #include statements are omitted from the code to save space.)

```
#define SIZE 5
int nums[SIZE] = {1, 2, 3, 4, 5};
int main {
    int i, total = 0;
    pid_t pid;
    pid = fork();
    if(pid == 0) {
        for(i = 0; i < SIZE; i++) nums[i] += i;
    }
    else if(pid > 0) {
        wait(NULL);
        for(i = 0; i < SIZE; i++) total += nums[i];
        printf("Total is %d\n", total);
    }
    return 0;
}
```

(3) Consider the following program. Each process produces outputs in a form of strings composed of letters. List all possible outputs printed on screen when executed. Also, justify your answers.

```
#include <unistd.h>
#include <sys/wait.h>
// W(A) means write(1, "A", sizeof "A"), which will display "A"
#define W(x) write(1, #x, sizeof #x)
int main() {
    W(A);
    int child = fork();
    W(B);
    if(child) wait(NULL);
    W(C);
    return 0;
}
```

5. Threads (5 points)

Explain the difference between concurrency and parallelism. Using these concepts, discuss how a multi-threaded application can run faster than a single-threaded application on a single-processor machine and on a multi-processor machine.

6. Process Scheduling 1 (20 points – 4 points each)

We have the following set of processes in the ready queue. The arrival time indicates the time when a process is first inserted into the back of the ready queue, and the burst time is the length of the burst. The unit of time is in milliseconds, and higher number indicates higher priority. We assume that the scheduler knows burst time of each process, and we ignore the overhead for switching processes.

Process	Arrival Time	Burst Time	Priority
P_1	0	40	10
P_2	10	20	30
P_3	20	12	50
P_4	30	6	20
P_5	40	18	40

(1) Waiting time is defined as the total time a process stays in the ready queue until the burst is finished. What is the average waiting time when we use FCFS? Explain why.

(2) What is the average waiting time when we use preemptive SJF?

(3) What is the average waiting time when we use preemptive priority scheduling?

(4) What is the average waiting time when we use RR with time quantum=12? Whenever a process runs for the duration of time quantum, the process exits the CPU and is inserted into the back of the ready queue, and the scheduler dispatches a process from the front of the queue.

(5) We define the **response time** as the time between arrival of the process and the time when the process is dispatched to the CPU for the first time. What is the average response time for FCFS, preemptive SJF, preemptive priority scheduling, and RR with time quantum=12? Which scheduling algorithm has the minimum average response time?

7. Process Scheduling 2 (10 points – 5 points each)

In priority-based scheduling, a process with low priority can result in starvation if processes with higher priority continue to arrive. In order to prevent starvation, we implement priority aging, where priority increases over time.

i) Each process has a priority value p_i which is initially 0 when the process is first admitted to the ready queue.

ii) When a process i is in the ready queue, its priority continuously increases with the speed of $\mathbf{s_i}$ per millisecond. (When a process is running on the CPU, p_i does not increase.)

iii) When the priority of a process reaches $\mathbf{p_{max}}$, the process preempts the running process and is dispatched to the CPU. If multiple processes reach $\mathbf{p_{max}}$ at the same time, the one with the highest $\mathbf{s_i}$ is scheduled first.

iv) If the CPU becomes idle, the process with the highest priority p_i is immediately dispatched to the CPU. If there are multiple processes with the highest p_i, the process with higher s_i is selected.

v) When a process running on the CPU is preempted and returns to the ready queue, its priority p_i becomes 0.

Suppose we have the following set of processes. We ignore the overhead for switching processes.

Process	Arrival Time	Burst Time	$\mathbf{s_i}$
P_1	0	20	5
P_2	0	15	40
P_3	0	10	20
P_4	0	25	10

(a) if p_{max} = 200, draw a Gantt chart that illustrates the execution of the processes.

(b) If we make p_{max} a very large number, the behavior of this algorithm becomes similar to which algorithm? Explain why.

8. Process Synchronization (15 points – 5 points each)

The bounded-buffer problem

We are going to implement a program where a producer and a consumer share an array that consists of 10 elements. The producer writes items to the buffer, whereas the consumer reads items from the buffer. Once a consumer reads an item from the buffer, the item is no longer used.

The implementation has three parts:

(i) initialization code for both producer and consumer

```
// shared variables
int items[10];
int iter_p = 0, iter_c = 0;
```

(ii) producer code

```
void producer(int nextp) {
  items[iter_p] = nextp;
  iter_p = (iter_p + 1) % 10;
}
```

(iii) consumer code

```
int consumer() {
  int nextc;
  nextc = items[iter_c];
  iter_c = (iter_c + 1) % 10;
  return nextc;
}
```

This code has three problems:

- The producer and consumer can access the shared array simultaneously which will result in incorrect results.

- If the buffer is empty (there is no produced item that is not consumed yet), the consumer could erroneously read an item.

- If the buffer is full (10 produced items are in the array), the producer could overwrite a previously produced item.

We want to prevent all three situations by adding semaphores to the code.

For example, we can use a binary semaphore as the following:

You can initialize a semaphore variable as follows:

```
semaphore s = 1;
```

This will declare a semaphore variable and initialize its value to 1. Then, you have two functions you can call using a semaphore variable.

```
wait(s);
signal(s);
```

Modify the implementation properly so that the three problems are removed (Hint: Use counting semaphores).

-------- End of the Exam -------