ID1206: Review Questions 1

Alexander Lundqvist

Fall 2022

1 Question 1

When a process creates a new process using the fork() operation, which of the following states is shared between the parent process and the child process? Stack, Heap, or Shared memory segments?

1.1 Answer

The new process only shares the shared memory segment and nothing else. The child process consists of a copy of the parents address space and receives a copy of the stack and heap.

2 Question 2

Explain what the output will be at LINE A.

```
int value = 15;
int main() {
    pid t pid;
    pid = fork();
    if (pid == 0) { /* child process */
        value += 10;
        return 0;
    } else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d",value); /* LINE A */
        return 0;
    }
}
```

2.1 Answer

The output will be 15. If we add a print command to the child process segment we can see that the child process prints 25. This is because fork() returns two values, 0 to the child process and the PID of the child process to the parent process.

```
int value = 15;
int main() {
  pid_t pid;
  pid = fork();
  if (pid == 0) { /* child process */
    value += 10;
    printf("Child process: value = %d", value); // Check the value in child process
    return 0;
  } else if (pid > 0) { /* parent process */
    wait(NULL);
    printf("Parent process: value = %d", value); /* LINE A */
    return 0;
  }
}
```

3 Question 3

How many processes are created in the following code?

```
int main() {
    fork();
    fork();
    fork();
}
```

3.1 Answer

8 processes are created. If we add a print statement after the last fork operation call we can see exactly how many processes gets executed. This is because after each fork, the child continues with the rest of the program after that fork, thus the amount of processes created can be described as $2^n = amount\ of\ processes$ and $n = amount\ of\ fork()\ calls$.

```
int main() {
    fork();
    fork();
    fork();
    printf("Process executed successfully!\n");
    return 0;
}
```

4 Question 4

See Section 4.1 of the Operating Systems Concepts book. Does the multi-threaded web server described in that section exhibit task or data parallelism?

4.1 Answer

The web server provides clients with web pages and page contents, and because it is multi-threaded it can serve several clients at a time. Since the web server performs the same task to every client, i.e, listen for requests and serve requests, and each process if performed on different subsets of the same data (that being the content on the server), we can say that it exhibits data parallelism.

5 Question 5

What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

5.1 Answer

User-level threads are easier to create and manage and can also run on any operating system since they aren't known by the kernel. While this sounds like user-level threads are better, kernel threads can take advantage of multiprocessing where as user-level threads can't.

6 Question 6

Describe the actions taken by a kernel to context-switch between kernel level threads.

6.1 Answer

Context switching is the act of storing the state of a process/thread so that it can be restored and continue with its execution at a later time. The main action taken by the kernel when context-switching between kernel level threads is that the value in the CPU registers of the thread being switched out gets saved. Then a new process gets selected and the CPU registers gets updated with the values of this new process.

7 Question 7

Explain the difference between preemptive and non-preemptive scheduling.

7.1 Answer

Preemptive scheduling is used when a process switches from a running state to a ready state or from a waiting state to a ready state which essentially means that a process may be paused and the control of the CPU can be allocated to a process that for example, could be deemed to have higher priority. Non-preemtive scheduling happens when a process switches from a running state to waiting state or when a process terminates. This means that the process controls the CPU until it has finished with its current CPU burst.

8 Question 8

Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use non-preemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
p_1	0.0	8
p_2	0.4	4
p_3	1.0	1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process P1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P1 and P2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.

8.1 Answer

a. This calculation is straight forward since FCFS dictates that the process arriving first is allowed control of the CPU.

$$\frac{(0+8-0.0)+(0+8+4-0.4)+(0+8+4+1-1.0)}{3} = 10.5333333333 \approx 10.53$$

b. In SJF, we look to the shortest burst time to find out which one to schedule next. We start with p_1 since it is the first and only in the queue. Next, when the queue fills up, we choose the next process that has the lowest burst time and so on. Since all processes will arrive in one time unit, we don't have to worry about choosing before the queue has filled up.

$$\frac{(0+8-0.0)+(0+8+1-1.0)+(0+8+1+4-0.4)}{3} = 9.53333333333 \approx 9.53$$

c. Since we will wait 1 unit of time, the queue will be completely filled and the algorithm will serve p_3 first, then p_2 and lastly p_1.

$$\frac{(1+1-1.0)+(1+1+4-0.4)+(1+1+4+8-0.0)}{3} = 6.86666666667 \approx 6.87$$

9 Question 9

Consider the following set of processes, with the length of the CPU burst time given in milliseconds. The processes are assumed to have arrived in

Process	**Burst Time**	**Priority**
p_1	2	2
p_2	1	1
p_3	8	4
p_4	4	2
p_5	5	3

the order P1, P2, P3, P4, P5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of these scheduling algorithms?

d. Which of the algorithms results in the minimum average waiting time (over all processes)?

9.1 Answer

a. Gantt charts of each scheduling algorithm.

Figure 1: CPU schedule diagram for FCFS algorithm.

Figure 2: CPU schedule diagram for SJF algorithm.

Figure 3: CPU schedule diagram for non-preemptive priority algorithm.

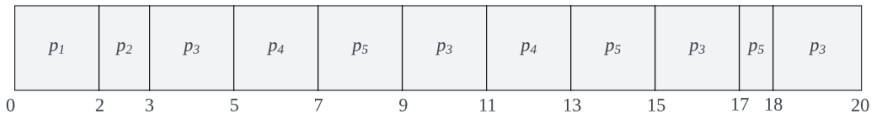

Figure 4: CPU schedule diagram for RR algorithm.

b. Turnaround time of each process for each algorithm.

Process	**FCFS**	**SJF**	**Prio**	**RR**
p_1	2	3	15	2
p_2	3	1	20	3
p_3	11	20	8	20
p_4	15	7	19	13
p_5	20	12	13	18

c. Waiting time of each process for each algorithm.

Process	**FCFS**	**SJF**	**Prio**	**RR**
p_1	0	1	13	0
p_2	2	0	19	2
p_3	3	12	0	12
p_4	11	3	15	9
p_5	15	7	8	13
Sum	31	23	55	36

d. We can see from the sum row in the previous table that the total wait time is lowest for SJF, hence the average will also be lowest.