Midterm Exam Reviews

ALL the assignments (A1, A2, A3) and Projects (P0, P1, P2) we have done so far.

Particular attentions on the following:

- System call, system kernel
- Thread/process, thread vs process in Linux
- differences between symmetric and asymmetric multiprocessing
- Java as an OOP: its threading, how to share data among threads.
- CPU scheduling algorithms: RR, priority, SJF, FCFS, Multilevel Queuing
- clustered systems vs. multiprocessor systems
- Source code in Project2.
- Shared memory vs. message passing
- essential properties of the following types of operating systems: (a) Interactive, (b) Time sharing, (c) Real time
- Pthread
- Microkernel
- major activities of an OS with regard to secondary-storage management
- PCB, context switching
- Mapping between user-level and kernel level threads: one-to-one, many-to-many, many-to-one
- general methods for passing parameters to the operating system
- Throughput, turnaround time, response time, CPU utilization
- he separation of mechanism and policy
- Virtual machine (VMWare)
- Stack, heap, shared memory segments.
- OS purposes
- OS' 2 modes of operation
- API, system calls
- Short-term, long-term, mid-term schedulers
- OS as a resource allocator
- major activities of an OS with regard to process management
- Interrupts, interrupt vector, trap
- DMA, how it works
- major activities of an OS with regard to memory management
- Starvation and aging

- **Understand the following code and their related questions in A2**.

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    /* fork a child process */
    fork();

    /* fork another child process */
    fork();

    /* and fork another */
    fork();

    return 0;
}
```

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid, pid1;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       pid1 = getpid();
       printf("child: pid = %d",pid); /* A */
       printf("child: pid1 = %d",pid1); /* B */
    }
    else { /* parent process */
       pid1 = getpid();
       printf("parent: pid = %d",pid); /* C */
       printf("parent: pid1 = %d",pid1); /* D */
       wait(NULL);
    }

    return 0;
}
```

Figure 3.29 What are the pid values?

- **Following A3 questions:.**

Explain the differences in the degree to which the following scheduling algorithms discriminate in favor of short processes:

a. FCFS
b. RR
c. Multilevel feedback queues

5.15 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context switching overhead is 0.1 millisecond and that all processes are long-running tasks. What is the CPU utilization for a round-robin scheduler when:

a. The time quantum is 1 millisecond
b. The time quantum is 10 milliseconds

4.10 Which of the following components of program state are shared across threads in a multithreaded process?

a. Register values
b. Heap memory
c. Global variables
d. Stack memory

4.12 As described in Section 4.6.2, Linux does not distinguish between processes and threads. Instead, Linux treats both in the same way, allowing a task to be more akin to a process or a thread depending on the set of flags passed to the clone() system call. However, many operating systems—such as Windows XP and Solaris—treat processes and threads differently. Typically, such systems use a notation wherein the data structure for a process contains pointers to the separate threads belonging to the process. Contrast these two approaches for modeling processes and threads within the kernel.

5.9 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

5.11 Consider the exponential average formula used to predict the length of the next CPU burst (Section 5.3.2). What are the implications of assigning the following values to the parameters used by the algorithm?

a. $\alpha = 0$ and $\tau_0 = 100$ milliseconds
b. $\alpha = 0.99$ and $\tau_0 = 10$ milliseconds

5.12 Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

a. Draw (**Hand drawing** OK) four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.
b. (**Using a Table**) what is the turnaround time of each process for each of the scheduling algorithms in part **a**?
c. (**Using a Table**) what is the waiting time of each process for each of the scheduling algorithms in part **a**?

5.15 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context switching overhead is 0.1 millisecond and that all processes are long-running tasks. What is the CPU utilization for a round-robin scheduler when:

a. The time quantum is 1 millisecond
b. The time quantum is 10 milliseconds

4.3 Describe the actions taken by a kernel to context-switch between kernellevel threads.
Answer: Context switching between kernel threads typically requires saving the value of the CPU registers fromthe thread being switched out and restoring the CPU registers of the new thread being scheduled.

5.2 Explain the difference between preemptive and nonpreemptive scheduling.
Answer: Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process. Nonpreemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.

5.4 What advantage is there in having different time-quantum sizes at different levels of a multilevel queueing system?
Answer: Processes that need more frequent servicing, for instance, interactive processes such as editors, can be in a queue with a small time quantum. Processeswith no need for frequent servicing can be in a queue with a larger quantum, requiring fewer context switches to complete the processing, and thus making more efficient use of the computer.

5.5 Many CPU-scheduling algorithms are parameterized. For example, the RR algorithm requires a parameter to indicate the time slice. Multilevel feedback queues require parameters to define the number of queues, the scheduling algorithms for each queue, the criteria used to move processes between queues, and so on.
These algorithms are thus really sets of algorithms (for example, the set of RR algorithms for all time slices, and so on). One set of algorithms may include another (for example, the FCFS algorithm is the RR algorithm with an infinite time quantum).What (if any) relation holds between the following pairs of algorithm sets?
a. Priority and SJF
b. Multilevel feedback queues and FCFS
c. Priority and FCFS
d. RR and SJF
Answer:
a. The shortest job has the highest priority.
b. The lowest level of MLFQ is FCFS.
c. FCFS gives the highest priority to the job having been in existence the longest.
d. None.