CS433 Operating Systems

Homework #2

Part II – Process Management

1. Describe the actions taken by a kernel to context-switch between processes.

(3 pts)

ANSWER: The OS will save the PC and user stack pointer for currently executing process, then it transfers the control to kernel clock interrupt handler. After this, the interrupt handler saves registers and machine state. Then, the OS invokes the scheduler to determine next process for execution. Lastly, the OS retrieves state of next process and restores register.

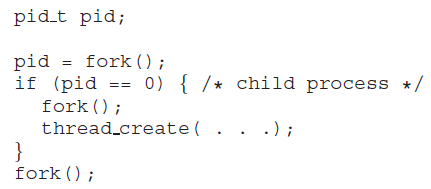
1. When a process creates a new process using the *fork( )* operation, which of the following state is shared between the parent process and the child process? (3 pts)
   1. Stack
   2. Heap
   3. Shared memory segments

ANSWER: C

1. Which of the following components of program state are shared across threads in a multithreaded process? (4 pts)
   1. Register values
   2. Heap memory
   3. Global variables
   4. Stack memory
2. Is it possible to have concurrency but not parallelism? Explain. (4 pts)

Yes, it is possible to have concurrency but not parallelism. Concurrency means multiple processes are progressing simultaneously whilst parallelism means multiple processes are running side by side (which can only be done if your CPU has more than one core) so, it is possible to have all tasks progress without having all tasks run simultaneously (which is parallelism).

1. Consider the following code segment: (2 pts)



* 1. How many unique processes are created? 6 processes
  2. How many unique threads are created? 2 threads

1. What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other? (6 pts) OS is not aware of User-level threads and they exist entirely within a process. OS is aware of kernel-level threads and kernel threads are scheduled by OS’s scheduler and requires a lightweight context switch to switch between.
2. Race conditions are possible in many computer systems. Consider a banking system with two methods: deposit(amount) and withdraw(amount). These two methods are passed the amount that is to be deposited or withdrawn from a bank account. Assume that a husband and wife share a bank account and that concurrently the husband calls the withdraw() method and the wife calls deposit(). Describe how a race condition is possible. (6 pts)

We can demonstrate this, for example, the balance in the account is 250.00 and the husband calls withdraw(50) and the wife calls deposit(100). The correct value should be 300.00. Since these two transactions will be serialized, the local value of balance for the husband becomes 200.00, but before he can commit the transaction, the deposit(100) operation takes place and updates the shared value of balance to 300.00. We then switch back to the 1 husband and the value of the shared balance is set to 200.00 - obviously an incorrect value.

1. Explain why spinlocks are not appropriate for single-processor systems yet are used in multiprocessor systems. (4 pts)

Spinlocks are not appropriate for single-processor systems because condition would break a process out of spinlock can be obtained only by executing a different process. If process doesn’t relinquish processor, other processes do not get opportunity to set program condition required for the first process to make progress..

.

1. A multithreaded web server wishes to keep track of the number of requests it services (known as **hits**.) Consider the following two strategies to prevent a race condition on the variable hits. The first strategy is to use a basic mutex lock when updating hits: (3 pts)

*int hits;*

*mutex lock hit lock;*

*hit lock.acquire();*

*hits++;*

*hit lock.release();*

A second strategy is to use an atomic integer:

*atomic t hits;*

*atomic inc(&hits);*

Explain which of these two strategies is more efficient.

The second approach is more efficient because the use of locks is overkill and requires a system call, as well as putting a process to sleep (resulting in a context switch). Automatic integer provides an automic update and will make sure no race condition happens on hits.

1. Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism — a spinlock or a mutex lock where waiting processes sleep while waiting for the lock to become available: (3 pts)
   1. The lock is to be held for a short duration. A spinlock would be faster than a mutex lock.
   2. The lock is to be held for a long duration. Mutex lock would be faster, as it allows the processor to schedule other processes.
   3. The thread may be put to sleep while holding the lock. Mutex lock because you don’t want waiting process to spin while waiting for other process to wake up.
2. Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs? (4 pts)

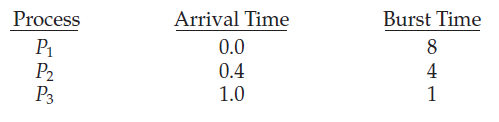
I/O bound programs have the property of performing only a small amount of computation before I/O. CPU bound programs may use up their entire time slice without performing any blocking I/O operations. So, I/O should be given higher priority as they do not use as much of their time slice compared to CPU bound programs.

.

1. Explain the difference between preemptive and non-preemptive scheduling. (4 pts)

Preemptive means that a process can be interrupted mid execution whilst nonpreemptive insures that a process only gives up CPU when it is done its CPU burst.

1. 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. (4 pts)



* 1. What is the average turnaround time for these processes with the FCFS scheduling algorithm? 10.533
  2. What is the average turnaround time for these processes with the SJF scheduling algorithm? 9.533

1. Consider two processes, *P*1 and *P*2, where *p*1 = 50, *t*1 = 25, *p*2 = 75, and *t*2 = 30. (8 pts)
   1. Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart?

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 | .. | 105 |

* 1. Illustrate the scheduling of these two processes using earliest deadline- first (EDF) scheduling.

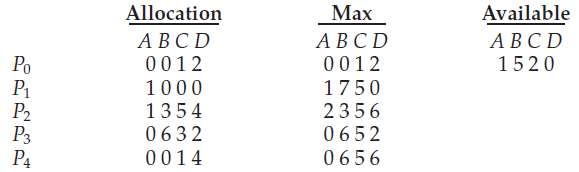
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P1 | P2 | P1 |  | P1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 25 | 55 | 80 | 110 | 135 | 150 |

1. Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock-free. (4 pts)

The system is deadlock free and can be shown by contradiction. If we suppose the system is deadlock, this implies each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must hold two resources. This process requires no more resource and will return resources when done.

1. Consider the following snapshot of a system: (6 pts)



Answer the following questions using the banker’s algorithm:

* 1. What is the content of the matrix **Need**?

The values of Need for processes P0 through P4 respectively are (0, 0, 0, 0), (0, 7, 5, 0), (1,0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).

* 1. Is the system in a safe state?

Yes.

1. Is it possible to have a deadlock involving only one single-threaded process? Explain your answer. (2 pts)

No, because deadlocking involves a circular hold and wait condition between two or more processes. This is best shown in the dining philosopher’s problem.