The University of Alabama in Huntsville

CPE 434/534 Operating Systems

**Solutions** Homework 1 and 2

Due Thursday, Sept 24th

Homework 1

1. What are the three main functional areas of an operating system. In addition to listing them explain what they are.

*The three main functional areas of an operating system are: time management, space management, and file management.*

1. Families of computer processors were introduced by IBM in the IBM 360. We still have processor families such as the X86, ARM, TI320Cxx, etc. Has the availability of ubiquitous operating systems like linux completely made the existence of processor families irrelevant. Explain your answer.

*Families of computer systems are still needed to describe the computing power of each element of the family. Thus although a 1 core and 4 core intel x86 computer have the same architecture and appear the same via linux, they are very different and offer very different capabilities.*

1. Text 1-14
2. A computer has a 4 state pipeline, each taking 1ns to complete its work. How many ips can it perform.

*While there are 4 stages, this neither speeds up the processing by a factor of 4 nor slows it down by a factor of 4. In the first second it could perform 999,999,997 instructions, but for all other times it would perform 1,000,000,000 instructions per second.*

1. Text 1-22
2. Can a write (..,…,nbytes) return any value other than nbytes

*Yes, it can return a smaller number of bytes if the output buffer is full and it can return -1 if an error occurs.*

1. Text 1-24

A 10 MB file is stored on a disk that transits track to track in 1 ms and has a rotational latency of 5 ms, and can transfer 200 MB/sec. If the head is currently on track 100, the desired data is on track 50, how long does it take to transfer the data.

*It takes 50ms to get to the correct track, 5 ms to get to the correct data, and 10/200=50ms to transfer data for a total of 105ms.*

Homework 2

1. Part of an operating system us usually written in the assembly language of the processor. Why? Explain your answer.

*Operating systems need access to specialized hardware registers such as the PC, Page Tables, etc. and there is no way to generate these instructions from C.*

1. If a process creates three threads and then forks.
   1. A- how many threads now exist

*There are two processes and three threads. Fork does not duplicate threads. Threads share heaps and this would not be possible between threads associated with different processes. Also, you need a handle for each thread so you can wait and/or terminate a thread. If you spawned new threads on a fork the identities of the new threads would be unknown.*

* 1. What do these threads share and what do they not share

*The three threads share what all threads share, text and heap but not stack. There are other things shared like open files and signal handlers but we have not discussed these yet and the previous answer is sufficient.*

* 1. How many user text heap and stacks now exist

*There are two user text sections, two user heap sections, and 5 user stack sections. Two of the stacks are associated with the processes and three are associated with the threads. Although processes have the same text heap and stack pages immediately after the fork for all intents they are different since as soon as you write to one the data is changed.*

* 1. How many kernel text heap and stacks now exist

*From the point of view of this class so far, there is one kernel text and one kernel heap section independent of the number of processes/threads. Thee will be 5 stack sections, one for each user process and/or thread. There are other potential stacks in the os that are used during interrupt handling but we have not really discussed these in detail and the previous answer is sufficient.*

1. We discussed user level and kernel level threads. Describe and explain one advantage and one disadvantage of each (a total of 4 answers)

*User threads: advantage- users can schedule them as they desire*

*User thread disadvantage: if a thread blocks the other threads can block*

*Kernel Threads: threads are scheduled against all other workload and will get more of a fare share of time.*

*Kernel thread disadvantage: they are expensive to start and manage in time.*

1. We have discussed spinwait, semaphores, pipes, and shared memory as a mechanism for synchronizing processes and threads. Provide one example problem where you would use each of these mechanisms.

*Spinwait should be used when the wait will be very small, microseconds, and other mechanisms, like semaphores, will take longer since they involve a system call. Radar signal processing pipeline is an example.*

*Semaphores should be used when multiple processes/threads need write access to some common piece of data and you need to provide mutual exclusive access to the data. This mighr be used by a producer/consumer application that has multiple processes entering and removing data from a shared data space.*

*Pipes allow point to point communication between processes in a single direction. An example of this would be a spell checker which checks words one at a time and then passes them to the next stage of processing.*

*Shared memory is not necessarily a synchronization mechanism but is used to share larger data sets between processes. It can be used for synchronization if the user puts semaphore like data into the shared memory and uses an indivisible operation, like SWAP, to set and test the flag.*

1. Write a very small program that uses 4 threads and global variables. Print out the addresses of a global variable in each thread that is and is not declared with the \_\_thread type.

*Sample Program (from Mr Alves)*

*#include <stdio.h>*

*#include <pthread.h>*

*#include <stdlib.h>*

*#include <unistd.h>*

*#define NUMBER\_OF\_THREADS 4*

*int varA = 5;*

*\_\_thread int varB = 3;*

*void \*showAddress(void \*tid)*

*{*

*printf("Thread #%d\n", (int)tid);*

*printf("varA address: %p\n, &varA);*

*printf("varB address: %p\n\n", &varB);*

*pthread\_exit(NULL);*

*}*

*int main()*

*{*

*pthread\_t threads[NUMBER\_OF\_THREADS];*

*int i, status;*

*for (i = 0; i < NUMBER\_OF\_THREADS; i++)*

*{*

*printf("Creating thread %d...\n", i);*

*status = pthread\_create(&(threads[i]), NULL,*

*showAddress, (void \*)i);*

*if (status != 0)*

*{*

*printf("Error creating thread #%d\n", i);*

*}*

*}*

*return(0);*

*}*

*Sample output*

*Creating thread 0...*

*Creating thread 1...*

*Thread #0*

*varA address: 0x804a02c*

*varB address: 0xb756cb3c*

*Creating thread 2...*

*Thread #1*

*varA address: 0x804a02c*

*varB address: 0xb6d6bb3c*

*Creating thread 3...*

*Thread #2*

*varA address: 0x804a02c*

*varB address: 0xb63ffb3c*

*Thread #3*

*varA address: 0x804a02c*

*varB address: 0xb5bfeb3c*

1. Text 2-21

*The problem is that the timer interrupt is managed by the operating system and may occur at any time during the user space activities, including when the user space thread library is switching threads. Because the scheduler might have data structures under revision at the time it may not be possible to do what the interrupt requires. The simplist solution to this problem is to have the interrupt handler note that the user thread library is in a critical section and set a flag rather than manipulate the structures. When the user level scheduler is finished with its critical section it should check to see if an interrupt occurred and then do the interrupt processing at that time.*

*A number of students suggested looking at the clock counter and postponing critical work if a clock interrupt was imminent. This solution seems to have come from somewhere but it is wrong in that from the time the clock timer is estimated the operating system might switch in and out a number of other processes so that the concept of time is not clear to the user level code.*

1. Text 2-45

*a round robin scheduling. The schedule looks like abcde abcde abde…. Two different answers are acceptable. The first one would assign the time of c terminating as 8 (when it actually terminated) and the other at 10, when the group of processes scheduled together ended their time slice. I accepted both. One was about 20 minutes and the other about 18.*

*B b priority scheduling each task is run to completion based on their priorities. So the schedule looks like b (pri 5), e (pri 4) a (pri 3) c (pri 2), d (pri 1) the completion times are 6, 14, 24, 26, 30. And the time is 20*

*C for first come first served each process is run to completion based on their arrival time, which is stated to be identical. So the execution order is a b c d e, the completion times are 10,16, 18, 22, 30 and the time is 19.2*

*D for shortest job first the processes run to completion and the execution order is c, d, b, e, a, the completion times are 2, 6, 12, 20, 30 and the answer is 14.*