**Homework 1**

**Released: Sep 26, 2024**

**Due date: Oct 3, 2024, 11:59pm**

**Total points: 100**

**Problem 1 (10 points)** What is the meaning of the term busy-waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answers.

Busy waiting:

When thread A requires resources(like lock, I/O) which is kept by another thread B. And thread A keep continuous checking in a loop. Which will cause a busy waiting, because the check instructions consume the CPU resources, CPU can not switch to other instructions. Like Spinlock

Other kind of waiting:

Sleep waiting: When a thread waiting for resources, instead of consuming CPU time, the thread is put to sleep; and the cpu can be allocated to others. like semaphores, mutex with sleep-wait behaviors.

Interrupt/Signal: the process or kernel sets up an interrupt or signal to notify it when the event happened.Which is commonly used in devices drivers, the hardware notifies the CPU that an operation is complete.

Solve busy waiting:

Busy-waiting can be minimized or avoid through more efficient synchronization techniques. Like semaphores, mutex with sleep-wait behaviors.

**Problem 2 (10 points)** Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.

1, Disabling interrupts in user-level programs may cause the system hangs when an infinite loop or deadlock occurs in the program.

2, Even if the user-level program execute correctly, disabling interrupts will cause the system to be unable to serve other requests, which will slow down the system, especially when the program need access I/O devices.

3, Not safe

**Problem 3 (10 points)** Given the following program that uses three memory segments in an address space as described in class (code segment, data segment, and stack segment):

char a[100];

main(int argc, char \*\* argv)

{

int d;

static double b;

char \*s = "boo", \* p;

p = malloc(300); return 0;

}

Identify the segment in which each variable resides and indicate if the variable is private to the thread or is shared among threads.

|  |  |  |  |
| --- | --- | --- | --- |
| variable | Which segment? | Shared among threads? | Reasons |
| a | Data segment | shared | Global variable |
| b | Data segment | shared | Static variable |
| d | Stack segment | private | Local variable within functions |
| char \*s= ‘boo’ | Pointer s is in Stack segment  “boo” is in Code segment | Pointer s is private  “boo” is shared | Code segment is shared by threads |
| char \*p=malloc(300) | Pointer p is in Stack segment  The memory allocated by malloc() is located in Heap segment | Point p is private  The memory allocated in the heap is shared |  |

**Problem 4 (10 points)** Which of the following instructions should be privileged:

|  |  |
| --- | --- |
| Instructions | Privileged |
| 1. Set the value of a timer | Yes |
| 1. Read the clock | No |
| 1. Clear menory | Yes |
| 1. Issue a trap instruction | No |
| 1. Turn off interrupts | Yes |
| 1. Modify entries in device-status table | Yes |
| 1. Switch from user to kernel mode | Yes |
| 1. access I/O device | Yes |

**Problem 5 (10 points)** Given the following piece of code:

main(int argc, char \*\* argv)

{

int child = fork(); int c = 5;

if(child == 0)

{

}

else

{

c += 5;

child = fork(); c += 10;

if(child)

{

c += 5;

}

}

}

How many different copies of the variable *c* are there? What are their values?

**Problem 6 (10 points)** 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:

* The lock is to be held for a short duration.
* The lock is to be held for a long duration.
* A thread may be put to sleep while holding the lock.

**Problem 7 (10 points)** A multithreaded web server wishes to keep track of the number of requests it services (known as ***hits***). Consider the two following strategies to prevent a race condition on the variable hits. The first strategy is to use a basic mutex lock when updating hits:

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.

**Problem 8 (10 points)** Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections at any point in time. As soon as N connections are made, the server will not accept another incoming connection until an existing connection is released. Illustrate how semaphores can be used by a server to limit the number of concurrent connections.

**Problem 9 (20 points)** Write a multithreaded program that outputs prime numbers. This program should work as follows: The user will run the program and will enter a number on the command line. The program will then create a separate thread that outputs all the prime numbers less than or equal to the number entered by the user.