Threads

4.1 Processes and Threads

1. Discuss the differences between the following:
   a. A process and a thread in terms of the following:
      - Overhead in creation & deletion.
      - Way of communication and its speed.

4.2 TYPES OF THREADS

2. Discuss the differences between the following:
   a. User-level and kernel-level threads in terms of the following:
      - Mapping.
      - Dealing with multi-processor systems.
      - Overhead on the kernel.
      - Portability.
      - Who is doing dispatching and scheduling?

3. Why all threads within a process will be blocked if a user-level thread calls a system call? Does that happen in kernel-level threads?

4. List the advantages and disadvantages of user-level threads over kernel-level threads and vice versa.

4.3 MULTICORE AND MULTITHREADING

5. What is the faster between the process switch and the thread switch? Explain why.

6. Would an algorithm that performs several independent calculations concurrently (for example matrix multiplication) be more efficient if it used threads or if it did not use threads? Discuss this point in both user/kernel threads and uni/multiprocessors.

4.4 WINDOWS PROCESS AND THREAD MANAGEMENT

7. Explain the states of threads in the windows operating system.

General Questions

8. A multiprocessor with eight processors has 20 attached tape drives. There is a large number of jobs submitted to the system that each require a maximum of four tape drives to complete execution. Assume that each job starts running with only three tape drives for a long period before requiring the fourth tape drive for a short period toward the end of its operation. Also assume an endless supply of such jobs.

a. Assume the scheduler in the OS will not start a job unless there are four tape drives available. When a job is started, four drives are assigned immediately and are not released until the job finishes. What is the maximum number of jobs that can be in progress at once? What are the maximum and minimum number of tape drives that may be left idle as a result of this policy?

b. Suggest an alternative policy to improve tape drive utilization and at the same time avoid system deadlock. What is the maximum number of jobs that can be in progress at once? What are the bounds on the number of idling tape drives?

9. Consider a multi-processor system and a multi-threaded program written using the many-to-many threading model. Let the number of user-level threads in the program be greater than the number of processors in the system.

Discuss the performance implications of the following scenarios:

a. The number of kernel threads allocated to the program is less than the number of processors.

b. The number of kernel threads allocated to the program is equal to the number of processors.

c. The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user-level threads.

10. Consider the following code using POSIX threads:

a. What does this program accomplish?

b. Here is the output from the executed program:

```
.o.o.o.o.oo.o.o.o.o.o.o.o.o.o..o.o.o.o.o
myglobal equals 21
```

Is this the output you would expect? If not, what has gone wrong?

```
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
int myglobal;
void *thread_function(void *arg)
{
    int i,j;
    for ( i=0; i<20; i++ )
    {
        j=myglobal;
        j=j+1;
        printf(".");
        fflush(stdout);
        sleep(1);
        myglobal=j;
    }
    return NULL;
}
int main(void)
{
    pthread_t mythread;
    int i;
    if ( pthread_create( &mythread, NULL, thread_function, NULL)
       )
    {
        printf("error creating thread.");
        abort();
    }
    for ( i=0; i<20; i++)
    {
        myglobal=myglobal+1;
        printf("o");
        fflush(stdout);
        sleep(1);
    }




    if ( pthread_join ( mythread, NULL ) )
    {
        printf("error joining thread.");
        abort();
    }
    printf("\nmyglobal equals %d\n",myglobal);
    exit(0);
}
```