CS 431 – Introduction to Computer Systems
Solutions - Mid semester exam 2005

1. Consider the code below. **(8 marks)**

```
public class Main {
        private ThreadA a;
        private ThreadB b;
   public Main() {
      a = new ThreadA();
      b = new ThreadB();
      a.start();
      b.start();
   }
   class ThreadA extends Thread {
      public void run() {
         setPriority(8);
         while (true) {
            System.out.println("Thread A is executing!");
         }
      }
   }
   class ThreadB extends Thread {
      public void run() {
         setPriority(7);
         while (true) {
            System.out.println("Thread B is executing!");
            try {
               sleep(5000);
} catch (Exception x) {}
         }
      }
   }
public static void main(String[] argv) {
     Main myMain = new Main();
```

```
        }
}
```

Assume that:

1. the system uses preemptive priority driven real-time kernel.
2. main thread will run at priority 5.

Assuming higher priority number means higher priority -

a). What will happen when the program is executed, and what is this situation called?

Ans.

Thread A never releases the CPU and thread B never gets a chance to execute.

b). Is it sufficient to just switch priorities between the two threads to avoid this situation?

Ans.

If the priorities are switched, thread A will get priority 7. Since the main program is executed in a thread with priority 5, the main program will get starved by thread A, when this thread is created. Thus, thread B will never be started and the problem persists.

2. The general behavior of the write-through cache coherence strategy was explained in the class, and the snooping cache idea was mentioned. Provide a pseudo code description of the operation of a snooping cache. **(5 marks)**

```
Each bus cycle:
do
    read data item A from the bus
    if A is being modified and is already in the cache
    update the value of the variable
done
```

3. Discuss the factors that must be considered in determining the degree of multiprogramming for a particular system. **(3 marks)**

Ans.

job scheduling algorithm, segmentation and paging (process scheduling, disk storage and memory management)

4. Draw a state diagram to represent the states and transitions for a user process.

(2 marks)

Refer to textbook.

5. Suppose that a machine's instruction set included an instruction named **swap** that operates as follows (as an indivisible instruction): **(4 marks)**

```
swap (a,b)
boolean a, b;
{
        boolean t;
        t ¬ a ;
        a ¬ b;
        b ¬ t;
}
```

Show how **swap** can be used to implement the P and V operations.

Ans.

```
do {
    key = TRUE; /* key is local boolean
     while ( key == TRUE)
         Swap (&lock, &key );
          //    critical section
       lock = FALSE;
       //      remainder section
} while (TRUE);
```

6. What is the effect of increasing the time quantum to an arbitrarily large number for round robin scheduling? **(2 marks)**

Ans.
It will be same as FCFS.

7. Suppose that the time to do a null RPC (i.e., 0 data bytes) is 1.0 msec, with an additional 1.5 msec for every 1K of data. How long does it take to read 32K from the file server in a single 32K RPC? How about as 32 1K RPCs? **(6 marks)**

Ans.

A single 32K RPC takes 1.5*32 + 1.0 = 49.0 msec

32 1K RPCs take 1.5*32 + 1.0*32 = 80.0 msec

8. In the class we described a multi-threaded server, showing why it is better than a single-threaded server and a finite-state machine server. Are there any circumstances in which a single-threaded server might be better? Give an example. **(4 marks)**

Ans.

A single-threaded server might be better than a multi-threaded server if the computations delegated to each thread are not independent of each other. A multi threaded application that spawns different thread to do computations and then uses these computed results only when all of them are available might be worser than the single threaded version of the same application. Also, if the length of the computation done by each thread is known a priori, a single process that prioritizes based on smallest computation time will perform better than a multi threaded solution.

9. In this problem you are to compare the time to read a file using a single-threaded server vs. multi threaded server. It takes 15 msec to get a request for work, dispatch it, and do the rest of the necessary processing, assuming that the data needed are in the cache. If a disk operation is needed, as is the case one-third of the time, an additional 75 msec is required, during which time the thread sleeps. How many requests/sec can the server handle if it is single threaded? If it is multi threaded? Explain your model. **(8 marks)**

Ans.

A single threaded server can handle 25 requests/second.

The multi threaded server can handle approximately (1000/75) requests/sec that need to access the disk. Since disk accesses happen every third request, the total number of requests is (1000/75)*3 = 40 requests/sec (approx)

10. Differentiate between **(2 marks each)**

a. multiprogramming and multiprocessing

Ans.

Multiprogramming: Multiple processes resident in the memory at a time OS decides which job to execute.

Multiprocessing: Systems with multiple processors sharing bus, clock (sometimes

memory and peripherals)
Either a master processor schedules the tasks (asymmetric) or each processor runs independent copy of OS and they communicate as needed.

b. security and protection

Ans.

Security: authentication of users to protect integrity of information stored in the system as well as physical resources. strictly internal problem
Protection: provide controlled access programs, processes or resources provided to the user. external environment needs to be taken into consideration.

c. deadlock prevention techniques and deadlock avoidance techniques

Ans.

Prevention techniques: ensure that at least one of the following conditions can not hold: mutual exclusion, hold and wait, no preemption, circular wait.
Low device utilization , reduced system throughput.
Avoidance techniques: prevent deadlock by restraining how requests can be made.
Less stringent, decision made on the basis of prior information about utilization of resources by the processes

d. a monolithic kernel and a microkernel

Ans.

Monolithic : conventional kernels.
better process and memory management.

Microkernel: remove all nonessential components from the kernel and implement them as system-level and user-level programs.
benefits: ease of extending OS, more security and reliability (as most processes run as user processes).
e.g.: Tru64 UNIX, QNX, Windows NT(hybrid structure)

11. What is the motivation behind the use of each of the following **(3 marks each):**

a. adaptive mutexes?

Ans.

An adaptive mutex is used to protect only those data items that are accessed by short

code segments. Spin waiting is used if a lock is held for less than a few hundred instructions.

b. CLUT (in a video card)?

Color Lookup Table is a hardware device that provides a mapping between pixel values and RGB color values.

12. Fill in the code that goes into the spaces marked by ... so that the critical section can be executed mutually exclusively using Test-and-Set over a shared boolean variable "lock", initialized to false. **(5 marks)**

```
while (true) {
        while ( ...
                ...
                 //    critical section
                 ...
                 //      remainder section
}
```

Ans:

```
while(true) {
        while(TestAndSet(lock));
        // critical section
        lock=false;
        // remainder section
}
```

13. Show how we can realize the semantics of synchronous send-receive communication between two processes using signals and waits? **(5 marks)**

Ans.

empty = N where N is the maximum number of messages that the shared memory can hold. It blocks when no more messages can be sent.

full = 0 blocks when there are no messages to receive.

Process Send(msg)	Process P2 Receive(msg)
P(empty);	P(full);

Enter Critical Section	Enter Critical Section
Write msg to shared memory location	Read msg from shared memory location
Exit Critical Section	Exit Critical Section
V(full);	V(empty);

Note: For the following questions, do not bother about being syntactically correct. If you do not remember the call syntax of a certain system call, make suitable assumptions about the argument/return type for the call.

14. Write a program which exits after any alphanumeric character is pressed on the keyboard. Program should consist of a process that creates a named pipe and reads characters from the keyboard, writing them to the pipe. It spawns a child process that checks for alphanumeric characters. **(4 marks)**

Solution is similar to programming assignment 1, problem 1.

15. In the programming assignment, you created a multi-client server using pthreads. In this question, you will need to create a multi-client server **without** using the fork/pthread calls. The idea behind this approach is to use a single process for all tasks: waiting for new connections, handling open connections and messages that are sent by the clients. This can be done using the select system call.

    Write a program for a multi-client server process. This server should be able to accept connections from several clients simultaneously and echo back to the client any message that it sends. **(15 marks)**

    The select system call is as follows:

```
int select(int numfds,
        fd_set *rd,
        fd_set *wr,
        fd_set *ex,
        struct timeval *timeout);
```

numfds - highest number of file descriptor to check.
r - set of file descriptors to check for reading availability.
wr - set of file descriptors to check for writing availability.
ex - set of file descriptors to check for exceptional condition.
timeout - how long to wait before terminating the call in case no file descriptor is ready.

select() returns the number of file descriptors that are ready, or -1 if some error occurred.

select() takes 3 sets of file descriptors to check upon. The sockets in the **rd** set will be checked whether they sent data that can be read. The file descriptors in the **wr** set will be checked to see whether we can write into any of them. The file descriptors in the **ex** set will be checked for exceptional conditions. Ignore this set for now, pass a NULL pointer instead of **ex**.

select() also takes an optional timeout value - if this amount of time passes before any of the file descriptors is ready, the call will terminate, returning 0 (no file descriptors are ready).

Assume **numfds** is passed as a command line argument to this program.

```
#include required header files
#define  BUFSIZE  <A fairly large number>
#define  PORT    <Some unique number > 1024>

void main()
{
        /* numfds was passed as command line argument */
        int             s, clientsock, clientsockaddrsize;
        char            buffer[BUFSIZE+1];
        struct sockaddr_in sa, clientsa;
        fd_set          rfd, waiting_rfd;

        /* initialise sa */
        /* create socket */
        s = socket(AF_INET, SOCK_STREAM, 0);
        if (s < 0) {
```

```
            perror("socket creation failed");
        }

        bind(s, (struct sockaddr *)&sa, sizeof(sa)); /* bind the
socket */

        /* listen for connections */
        listen(s, 10);


        /* socket s is open to receive new incoming connections. */
        FD_ZERO(&rfd); /* clear the set of fds with address &rfd */
        FD_SET(s, &rfd); /* add s to the set of read file descriptors */

        while (true) {

            waiting_rfd = rfd;
            rc = select(numfds, &waiting_rfd, NULL, NULL, NULL);

          /* we know that a new connection request arrived when
            's'is ready for reading . */
            if (FD_ISSET(s, &waiting_rfd)) {
                /* accept connection from the client */
                clientsock = accept(s, (struct sockaddr *)&clientsa,
&clientsockaddrsize);

                /* add new socket to the read file descriptor set */
                FD_SET(clientsock, &rfd);

                continue;
            }

            /* find the sockets that are ready for reading, */

            for (i=0; i < numfds; i++) {
                if (i != s && FD_ISSET(i, &waiting_rfd)) {
```

```
                    retcode = read(i, buf, BUFSIZE); /* read */

                    /* client closed the connection*/
                    if (retcode == 0) {
                        close(i);/* close the socket */
                        FD_CLR(i, &rfd);
                    }

                    else {
                        /* echo data back to the client */
                        write(i, buffer, retcode);
                    }
                }
            }
        }
}
```

16. Explain the crux of the software problem that caused the total system reset of the Pathfinder spacecraft. **(5 marks)**

Pathfinder contained an "information bus" which can be thought of as shared memory. Access to it was synchronized by the use of mutexes. The real source of problem involved 3 tasks in the system. A high priority "Bus Management" task (A) that ran frequently, a low priority "Meteorological Data Gathering" task (C) which ran infrequently and used the bus to save/send the gathered data, and a medium priority communications task (B).

Whenever task C wanted to send data it would lock the information bus using the mutex, write on the bus and release. The problem occurred when task C locked the bus to write some data, task A would be in the ready queue to get scheduled. Since it also would need to lock the "Bus" mutex, it would block as task C had already locked it. Then before task C could release the mutex, task B would also be ready to be scheduled. Task B being higher priority than task C would get scheduled preempting task C. Then task B would execute for long duration and would result in a watchdog timer expiry and hence system reset.
This is a classical example of priority inversion problem.

17. Complete the following sentences to make them TRUE: **(2 marks each)**

a) The rate monotonic policy assignment scheme (RMA) allocates high priorities to tasks with **low periods.**

b) A set of n tasks with calculation times Ci and periods (and relative deadlines) Ti are not schedulable under EDF if

c) RMA is preferred by practitioners because it is a static priority algorithm.

d) Priority inheritance cannot avoid chained blocking because
of transitive inheritance (as in the following situation):
Suppose a medium priority thread attempts to take a mutex owned by a low priority thread, but while the low priority thread's priority is elevated to medium by priority inheritance, a high priority thread becomes runnable and attempts to take another mutex already owned by the medium priority thread. The medium priority thread's priority is increased to high, but the high priority thread now must wait for both the low priority thread and the medium priority thread to complete before it can run again.

e) One thing I do not like about this course is