University of California, Berkeley
College of Engineering
Computer Science Division – EECS

Fall 2017 Ion Stoica

First Midterm Exam

September 28, 2017
CS162 Operating Systems

Your Name:	
SID AND 162 Login (e.g. "s042"):	
TA Name:	
Discussion Section Time:	

General Information:
This is a **closed book and one 2-sided handwritten note** examination. You have 80 minutes to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points for that question. You should read **all** of the questions before starting the exam, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. ***Make your answers as concise as possible.*** If there is something in a question that you believe is open to interpretation, then please ask us about it!

Good Luck!!

QUESTION	POINTS ASSIGNED	POINTS OBTAINED
1	22	
2	22	
3	20	
4	21	
5	15	
TOTAL	100	

P1 (22 points total) True/False and Why? **CIRCLE YOUR ANSWER**. For each question: 1 point for true/false correct, 1 point for explanation. An explanation cannot exceed 2 sentences.

a) The order in which you declare the members of `struct thread` in Pintos matters.

TRUE FALSE

Why?

b) If one thread closes a file descriptor, then it will be closed for all threads in the system, regardless of the process they are belonging to.

TRUE FALSE

Why?

c) The only way a thread can modify another thread's stack variable is if the thread is passed the address of that variable as an argument during creation.

TRUE FALSE

Why?

d) A binary semaphore (i.e., a semaphore which can only take values 0 and 1; if the semaphore is 1 and you increment it, its value remains 1) is semantically equivalent to a lock.

TRUE FALSE

Why?

e) When you call `fputs(string, outfile)`, the content of "string" is immediately written to "outfile" on disk.

TRUE FALSE

Why?

f) The function `pthread_yield()` will always ensure that another thread gets to run.

TRUE FALSE

Why?

g) Hardware interrupts can be used to implement locks.

TRUE FALSE

Why?

h) When a thread is performing an I/O operation, such as `read()`, the thread is busy-waiting until the operation completes.

TRUE FALSE

Why?

i) When a hardware interrupt occurs, the kernel enqueues the interrupt and serves it when the current running thread completes.

TRUE FALSE

Why?

j) With a monitor, the program can call `wait()` in the critical section on a condition variable associated to that monitor.

TRUE FALSE

Why?

k) Context switching between two threads belonging to the same process is less expensive than context switching between two threads belonging to two different processes.

TRUE FALSE

Why?

P2 (22 points) **Re: Zero – Starting Life in Another Process:** Consider the following C program. Assume that all system calls succeed when possible.

```
void* rem(void *args) {
      printf("Blue: %d\n", *((int*) args));
      exit(0);
}
void* ram(void *args) {
      printf("Pink: %d\n", ((int*) args)[0]);
      return NULL;
}
int main(void) {
      pid_t pid; pthread_t pthread; int status; //declaring vars

      int fd = open("emilia.txt", O_CREAT|O_TRUNC|O_WRONLY, 0666);
      int *subaru = (int*) calloc(1, sizeof(int));
      printf("Original: %d\n", *subaru);

      if(pid = fork()) {
            *subaru = 1337;
            pid = fork();
      }
      if(!pid) {
            pthread_create(&pthread, NULL, ram, (void*) subaru);
      } else {
            for(int i = 0; i < 2; i++)
                  waitpid(-1, &status, 0);
            pthread_create(&pthread, NULL, rem, (void*) subaru);
      } pthread_join(pthread, NULL);

      if(*subaru == 1337)
            dup2(fd, fileno(stdout));
      printf("All done!\n");
      return 0;
}
```

a) (4 points) Including the original process, how many processes are created? Including the original thread, how many threads are created?

b) (6 points) Provide all possible outputs in standard output. If there are multiple possibilities, put each in its own box. You may not need all the boxes.

c) (4 points) Provide all possible contents of `emilia.txt`. If there are multiple possibilities, put each in its own box. You may not need all the boxes.

d) (4 points) Suppose we deleted line 28, how would the contents of `emilia.txt` change (if they do)?

e) (4 points) What if, in addition to doing the change in part (d), we also move line 12 (where we open the file descriptor) between lines 19 and 20? What would emilia.txt look like then?

P3 (20 points) **Synch Art Online – Ordinal Scale:**

a) (8 points) [Spinlock using swap] In lectures, you learnt how to implement locks using test & set. Your task here is to implement locks using the *swap* primitive. To recapitulate, swap has the following semantics, and is executed *atomically*:

```
void swap(int *a, int *b) {
      int temp = *a;
      *a = *b;
      *b = temp;
}
```

You have to implement Initialize, Acquire and Release for the lock operations. It is *OK* to busy wait.

```
void Initialize(int* lock) {


}

void Acquire(int* lock) {




}

void Release(int* lock) {


}
```

b) (8 points) [Sleeping Barber Problem] A barber shop has one barber chair, and a waiting room with **N** chairs. The barber goes to sleep if there are no customers. If a customer arrives, and the barber chair along with all **N** chairs in the waiting room are occupied, he leaves the shop. Otherwise, the customer sits on the waiting room chair and waits. If the barber is asleep in his chair, the customer wakes up the barber.

Consider that the barber and each of the customers are independent, concurrent threads. Fill in the following blanks to ensure that the barber and the customers are synchronized and deadlock free. Assume each semaphore has P() and V() functions available as semaphore.P() and semaphore.V():

```
Semaphore barberReady = 0;
Semaphore accessWaitRoomSeats = 1;
Semaphore customerReady = 0;
int numberOfFreeWaitRoomSeats = N;

void Barber () {
 while (true) {


   numberOfFreeWaitRoomSeats += 1;

   cutHair();                 // Cut customer's hair

 }
}

void Customer () {

 if (numberOfFreeWaitRoomSeats > 0) {
   numberOfFreeWRSeats -= 1;



   getHairCut();              // Customer gets haircut :)
 } else {

   leaveWithoutHaircut();           // No haircut :(
 }
}
```

c) (4 points) Describe a scenario where a customer can get starved, i.e., get stuck waiting in the waiting room forever. Briefly outline a way to fix this.

P4 (21 points total) **Networking is an Important Skill!:** The code below implements a server that handles multiple connections. (We ignore disconnections and other socket errors, as well as fork() failure. You could also ignore those failures when answering the following questions.)

```
0  int pid = getpid()
1  bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr))
2  listen(sockfd,5);
3  while (1) {
4    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,
&clilen);
5    if (fork() == 0) {
6       // do some communication;
7       exit(0);
8    } else {
9       if (some condition) break;
10   }
11 }
```

a) (6 points) [Close Sockets] Insert code to close sockets whenever appropriate. You might need to insert multiple close() statements. You should close a socket as soon as it's not needed. You will not receive point for a delayed close().

Example: After line 1: close(sockfd) # note this example might be incorrect

b) (6 points) [Send] Assume we permit at most 3 open connections from clients, at ANY time. One way to do this is to stop accepting new connections when there are already 3. But this leaves a waiting client in the dark. We want to inform the client that "system is overloaded". To do that, when the 3rd connection is accepted, we tell the client that "system is overloaded; reconnect later." and then immediately close that connection afterwards. (In this way, we have only 2 working connections but this is fine). Fill in the blanks on the following page to complete this code. You don't need to close sockets for this problem.

```
int count = 0;
int status;
while (1) {
   while (count > 2) {


   }
   newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,
&clilen);
   count++;

   if (fork() == 0) {



         } else {
               # do some communication
         }
         exit(0);
   } else {
         if (some condition) break;
   }
}
```

c) (6 points) [Read] Back to the original code. Now assume the parent process with the listening socket should be terminated whenever ANY of the open connections receive a character 'q' from clients. (The existing open connections should still be working.) We expand line #6 to do that. Fill in the blanks to complete this code.

```
char reqbuf[MAXREQ];
while (1) {






}
```

d) (3 points) What happens to the child processes when the listening process in c) is terminated? Are the children going to be terminated as well? If yes, provide a design so that child processes will be not terminated.

P5 (15 points) **Back to Our Scheduled Programming:** Consider three processes P1, P2, and P3, being scheduled by a preemptive scheduler. Assume each process has a single kernel thread, and the context switching between two processes is 100 usec, while the time slice is 10ms.

a) (5 points) Assume that none of the processes perform I/O operations or waits for a signal, i.e., the kernel context switch a process only when its time slice expires (using the timer interrupt). What is the scheduling overhead of the system, i.e., the total context switching time over the total time the CPU is busy?

b) (5 points) Assume that one process is performing an I/O every 1 ms after it is scheduled, and the scheduler is using round-robin. What is the scheduling overhead of the system now?

c) (5 points) Now assume that all processes share a critical section that takes much longer than a times slice, and they use busy-waiting to implement mutual exclusion. Assume that at any given time there is one process in the critical section, and the scheduler is using the round-robin discipline. What is the percentage of time the CPU is doing useful work (the only useful work in this case is performed in the critical section.)