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

Fall 2018 Ion Stoica

First Midterm Exam

October 1, 2018
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
P1	20	
P2	20	
P3	10	
P4	30	
P5	20	
TOTAL	100	

NAME: ____

P1. (20 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) Ignoring context switching overhead, round robin scheduling always has better average response time than first-come-first-serve

TRUE FALSE

Why?

b) Context switches are always involuntary

TRUE FALSE

Why?

c) Every thread has its own heap

TRUE FALSE

Why?

d) Using the stream API, it is possible to read from a random location of a file

TRUE FALSE

Why?

e) In Pintos, the default scheduler implements First-Come, First-Serve

TRUE FALSE

Why?

NAME: ____

f) Assume a process opens a file, stores the file descriptor in a variable `fd`, and then calls `fork()`. If the parent closes `fd`, the child's `fd` wil be closed as well

TRUE FALSE

Why?

g) Starvation implies a system is deadlocked

TRUE FALSE

Why?

h) Semaphores can be used to implement scheduling constraints

TRUE FALSE

Why?

i) Implementing critical sections and mutual exclusion involves waiting

TRUE FALSE

Why?

j) In Pintos, the ordering of members inside the thread struct does not matter

TRUE FALSE

Why?

NAME: ____

P2. (20 points total) What The Fork.

Eleanor, a CS162 student, wants to create a multithreaded program that she can use to tell everyone how much she loves her favorite operating systems class. She wrote the program below, but she finds that different runs produce different outputs.

```
void* func1(void* args) {
    printf("is\n");
    return NULL;
}

void* func2(void* args) {
    printf("CS162\n");
    return NULL;
}

int main(void) {
 pid_t pid;
 pthread_t pthread;
  int *ret = (int*) malloc(sizeof(int));
  int status;
  pid = fork();
  if (!pid) {
      pthread_create(&pthread, NULL, func2, (void*) ret);
        pthread_join(pthread, NULL);
  } else {
      printf("the\n");
  }
  printf("best!\n");
  return 0;
}
```

a) (**10 points)** List all of the possible outputs that her program could display when run. Assume that calls to fork and pthread_create always succeed.

List all possible outputs, one in each column.

NAME: ____

b) **(10 Points)** Modify Eleanor's program such that the output will always be:

```
CS162
is
the
best!
```

Fill in the blanks to show your changes to the original program.

- You must use `pthread_create` at least once.
- You may not call `printf` or the helper functions (`func1`/`func2`) directly.
- Write at most one statement per line. You may not need all lines

```
void* func1(void* args) {
   printf("is\n");
   return NULL;
}
void* func2(void* args) {
   printf("CS162\n");
   return NULL;
}
int main(void) {
   pid_t pid;
   pthread_t pthread;
   int *ret = (int*) malloc(sizeof(int));
   int status;

   ______________________________________
   pid = fork();
   if (!pid) {
         _____________________________________
         pthread_create(&pthread, NULL, func2, (void*) ret);
         pthread_join(pthread, NULL);

         _____________________________________

         _____________________________________

         _____________________________________
   } else {

         _____________________________________
         printf("the\n");

         _____________________________________
   }

   _____________________________________
   printf("best!\n");
   return 0;
}
```

NAME: ____

P3. (10 points total) Networking

The following program implements a simple network server that accepts client connections and writes back the string "`Hello!`", and then closes the connection. Assume no failures occur in any function calls and that reads and writes to sockets will always result in the entire message being received or sent.

```
#define BUFFER_SIZE 4096

int main() {
    struct sockaddr address;
    int addrlen = sizeof(address);
    /* socket initialization code omitted */
    int serverfd = socket(PF_INET, SOCK_STREAM, 0);
    bind(serverfd, &address, sizeof(address));
    listen(serverfd, 5);
    char buffer[BUFFER_SIZE];
    char *string = "Hello!";
    while (1) {
        int newsockfd = accept(serverfd, &address, &addrlen);
        write(newsockfd, string, strlen(string));
        close(newsockfd);
    }
    close(serverfd);
    return 0;
}
```

a) **(5 points)** On receiving the message, the server writes it to an existing log file in the local directory named "log.txt". Assume that messages are no larger than 4096 bytes. Fill in the code below to accomplish this task. Please write at most one statement per line.

```
char buffer[BUFFER_SIZE];
char *string = "Hello!";
int fd = open("log.txt", O_WRONLY | O_APPEND);
while (1) {
    int newsockfd = accept(serverfd, &address, &addrlen);

    ___________________________________________________

    ___________________________________________________
    write(newsockfd, string, strlen(string));
    close(newsockfd);
}
close(fd);
```

b) **(5 points)** Instead of writing to the log, we want to print messages to the console. Add only a single statement to your solution in part a) in order to print client messages to standard output instead of to the log. You may assume that part a) was implemented correctly, and you may not modify any of part a).

```
while (1) {
    int newsockfd = accept(serverfd, &address, &addrlen);

    _____________________________________________________
    /* solution to part a */
    write(newsockfd, string, strlen(string));
    close(newsockfd);
}
```

NAME: ______________________________________

P4. (30 points total) CPU Scheduling.

Consider the following single-threaded processes, and their arrival times, CPU bursts and their priorities (a process with a higher priority number has priority over a process with lower priority number):

Process	Burst Time	Arrival Time	Priority
A	3	1	1
B	3	2	3
C	4	2	2
D	1	4	4

You will consider three scheduling algorithms: (1) round robin (RR), (2) shortest remaining time first (SRTF), and (3) priority scheduling. The following apply:

- All schedulers are preemptive.
- The time quanta of the RR scheduler is 1.
- If a process arrives at time x, it can be scheduled immediately.
- If two processes arrive at the same time, they are inserted in the ready queue in the lexicographical order. For example, if B and C arrive at the same time, B is inserted first, and C second in the ready queue.
- In the case of the RR scheduler, a new arriving process is inserted at the *end* of the ready queue. When the RR quantum expires, the currently running thread is added at the end of the ready list before any newly arriving threads.
- The RR scheduler ignores the process priorities.
- The SRTF scheduler uses priorities to break ties, i.e., if two processes have the same remaining time we schedule the one with the highest priority.

NAME: ________________________________

a) **(15 points)** Given the above information, fill in the following table. You may not need all the spaces.

Time	Round Robin	SRTF	Priority
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
Average completion time			

Duplicate of table for your convenience:

Process	Burst Time	Arrival Time	Priority
A	3	1	1
B	3	2	3
C	4	2	2
D	1	4	4

NAME: ____

b) **(15 points)** Additionally, now assume there are two locks L1 and L2, respectively. Furthermore assume the following:

- Process A acquires L1 and L2 in its first unit of time. It releases L1 in its last unit of time and releases L2 in its second unit of time
- Process B acquires L1 in its first unit of time and releases L1 in its last unit of time
- Process D acquires L2 in its first unit of time and releases L2 in its last unit of time
- Processes will sleep if acquiring lock fails. (Still scheduled, does not consume a burst)
- Priority donation is implemented.

Given the above information, fill in the following table. You may not need all the spaces. If it helps, you may specify when a process is put to sleep/blocked

Time	Round Robin	SRTF	Priority
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
Average completion time			

NAME: ____

P5. (20 points total) Concurrency.

Consider the following piece of code, which can be executed by one or more threads:

```
x = x + 1
y = x + y
```

Assumptions:

- Before any thread starts running, both x and y are initialized to 1. After that x and y are *only* updated by the threads executing the above code segment.
- A thread can be preempted at any point during its execution.
- Each of the two instructions is atomic.

a) **(5 points)** Assume two copies of the above piece of code run concurrently in two threads. Write down all possible values after both threads finish, one per column (number of outputs might be smaller than the number of columns).

x								
y								

b) **(5 points)** Same question, but now assume that three copies of the code segment in three threads.

x								
y								

c) **(5 points)** Assume *10* copies of the same program run concurrently. What is the *maximum* value of y?

d) **(5 points)** Assume *10* copies of the same program run concurrently. What is the *minimum* value of y?

NAME: ____

P6. (0 points) Potpurri

a) I am glad that there aren't anime references in this exam

TRUE TRUE

Why?

b) Name the one pop culture reference in this exam

c) Draw your TA