Name: Qixiang Liu 2856114

Put all answers on this sheet. Show your work (this is required for full credit and helpful for partial credit). Good Luck!

1. Choose best matching terms to complete the following sentences. (7 points)

   A. Batch processing ✓?
   B. Time sharing
   C. Micro
   D. Monolithic
   E. Process
   F. Thread
   G. Orphan
   H. Multiprogramming ✓
   I. Zombie
   J. Race condition
   K. Kernel
   L. User
   M. Message passing
   N. Shared memory
   O. Deadlock

D 1) Linux's structure is considered as a ___D___ kernel architecture.

2) System calls are executed in ___K___ mode.

B 3) In a ___N___ system, the OS frequently switches tasks to improve interactivity of the system. (i.e., better responsiveness)

4) The ___M___ based Inter-Process Communication (IPC) typically incurs high communication overhead.

5) A ___E___ has its own dedicated address space.

I 6) If a child process finishes but its parent does not wait (by calling wait system call), the child process becomes a ___G___ process as it cannot be fully removed from the OS.

7) ___J___ is a situation when two or more threads read and write shared data at the same time and the correctness depends on the order of execution.

2. State whether the following statements are true (T) or false (F). (13 points)

F T 1) CPU cache is managed by the OS. CPU cache is hardware, OS is software

T. F 2) Hardware timer and interrupt support are necessary to implement preemptive CPU schedulers in OS. 1. Timer and interrupt is hardware
2.中断必须打断当前操作 -preemptive

T 3) A software interrupt is generated when a system call is called

T 4) On a context switch, OS saves the current CPU register values (e.g., program counter, stack pointer, and general-purpose registers) to the current process's process control block (PCB).

T 5) In a single-core processor, only one process in the system is in the 'running' state.

9 F 6) Multiple threads can share a stack.

F 7) The anonymous pipe is a shared memory based Inter-Processor Communication (IPC) mechanism.

F 8) When the OS natively supports threading at the kernel level, if any one of threads blocks on a system call, all the other threads will also block as well.

F 9) Multi-threaded software architecture offers stronger protection/isolation than multi-process software architecture.

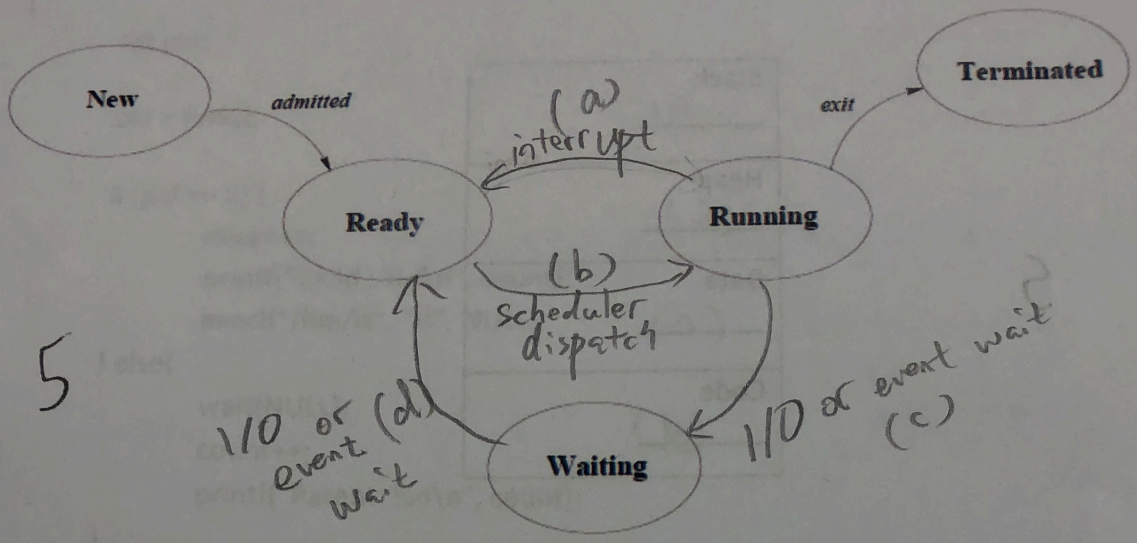
T 10) Spinlock is an effective synchronization mechanism to protect short critical sections

T F 11) If a resource allocated to a process can be preempted, deadlock cannot happen.
not preempted...

F 12) Round-robin (RR) with a small quantum size is a good scheduling policy to maximize throughput.

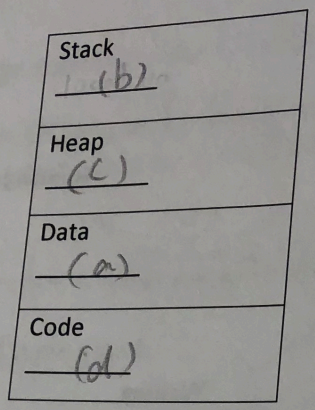
F. T 13) Shortest-remaining-time-first (SRTF) is a non-preemptive scheduling algorithm.
preemptive?

3. [Process] The process state diagram below shows all the possible process states, but only two process transitions. Complete the figure by illustrating how the process changes states on the following events: (a) interrupt, (b) scheduler dispatch, (c) I/O or event completion, and (d) I/O or event wait. (5 points)



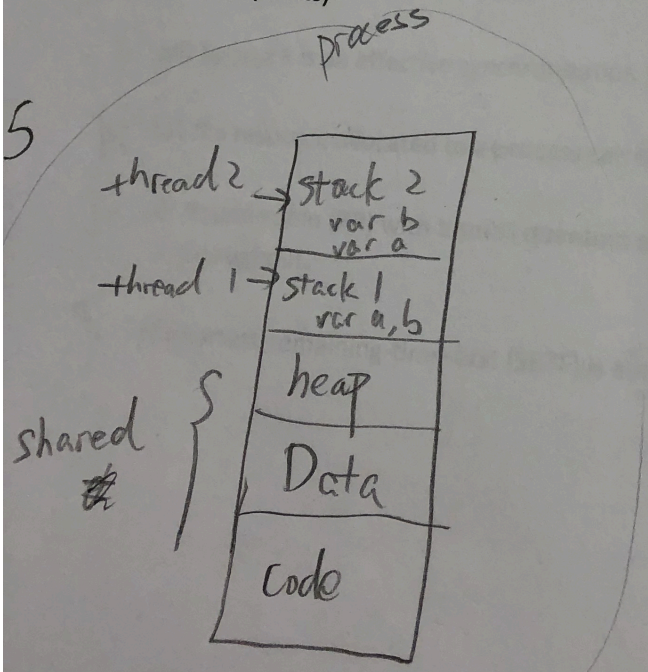
4. [Process] The following figure shows ... following components of your high-level C program resid... global variables, (b) local variables, (c) dynamically allocated memory, (d) program binary (5 points)

5



5. [Threads] Draw the address space of a process with two threads. Indicate all regions of the process. (5 points)

5



6. [Process] Consider the following program. (10 points)

```
int count = 0;
int main()
{
        Int pid;

        pid = fork();

        if (pid == 0) {
                count++;
                printf("Child : %d\n", count);
                execl("/bin/ls", "ls", NULL);
        } else{
                wait(NULL);
                count++;
                printf("Parent: %d\n", count);
        }
        count++;
        printf("Main: %d\n", count);
        return 0;
}
```

You may assume that all system calls (fork and execl) are successful. Ignore any other synchronization issues. The output from the program "/bin/ls" can just be indicated as: ''Output from ls''.

(a) Is 'printf()' a system call or a library call? (1 point) library call -

(b) is 'fork()' a system call or a library call? (1 point) system call ✓

(c) What output lines are going to be printed on screen after executing the program? (8 points)

8

Child: 1 ✓
Output from ls ✓
~~Main: 2~~ -2
Parent: 1 -
Main: 2 ✓

7. [Synchronization] complete the following (10 points)

(a) complete the pseudo-code definition of TestAndSet instruction. (4 points)

```
Int TestAndSet(int *lock)
{
        Int ret = *lock;
        *lock = 1;
        return ret;
}
```

(b) Complete spinlock_lock() implementation using the TestAndSet instruction. (3 points)

10

```
Void spinlock_init(int *lock)
{
        *lock = 0;
}

void spinlock_lock(int *lock)
{
        While (TestAndSet(lock))
                ;
}
```

(c) Complete spinlock_unlock() implementation (3 points)

```
void spinlock_unlock(int *lock)
{
    *lock = 0;
}
```

8. [Synchronization] Consider the following C program. (10 points total)

```
int count = 5;
pthread_mutex_t lock;

void worker1()
{
      count = count + 1;
}

void worker2()
{
      count = count - 1;
}

int main()
{
      pthread_t tid[2];
      pthread_mutex_init(&lock, NULL);

      pthread_create(&tid[0], NULL, worker1, NULL)
      pthread_create(&tid[1], NULL, worker2, NULL)

      pthread_join(tid[0], NULL);
      pthread_join(tid[1], NULL);

      printf("Main: %d\n", count);
      return 0;
}
```

Race condition
Lab content

(a) What are the possible outputs? (5 points)

5, 4, 6 / (No sychron!)

(b) The following shows prototypes of pthread mutex API. Remove the race condition by using the mutex API on the code above. (5 points)

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

```
void worker1( ){
pthread_mutex_lock(&lock);
count = count+1;
pthread_mutex_lock(&lock);
}
```
unlock

```
void worker2( ){
pthread_mutex_lock(&lock);
count = count+1;
pthread_mutex_lock(&lock);
}
```
unlock

9. [Synchronization] Consider

	Monitor version	Semaphore version
1	Mutex lock;	Semaphore mutex = 1;
2	Condition full, empty;	Semaphore full = 0;
3		Semaphore empty = N;
4	void produce (item)	
5	{	void produce (item)
6	lock.acquire( ) ✓	{
7	while (queue.isFull()) {	P(&empty); ✓
8	empty.wait(&lock); ✓	P(&mutex);
9	}	queue.enqueue(item);
10	queue.enqueue(item);	V(&mutex);
11	full.signal(); ✓	V(&full); ✓
12	lock.release(); ✓	}
13	}	Item consume()
		{
14	Item consume()	P(&full); ✓
15	{	P(&mutex);
16	lock.acquire();	item = queue.dequeue();
17	while (queue.isEmpty()) {	V(&mutex);
18	full.wait(&lock); ✓	~~V(&full);~~ ✓ V(&empty)
19	}	return item;
20	item = queue.dequeue(item);	}
21	empty.signal();;	
22	lock.release();	
23	return item;	
	}	

10

Complete the code (10 points).

...ding buffer problem. (10 pt.)

10. [Deadlock] Consider the following initial condition. (10 points)

There are three types of resources R1, R2, and R3. The following table shows the how many instances of each resource type are currently available.

R1	R2	R3
1	1	2

Avail

In the system, there are four processes, from P1 to P4, and the following table shows each process's current resource allocation and its maximum need to be able to terminate. For example, P1's maximum need is 3 instances of R3 and that it currently has only one R3 instance. Therefore, P1 would need additional two instances of R3 to be able to terminate.

	Current Allocation			Max			Still Need		
Process	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	1	0	1	1	0	3	0	0	2
P2	2	0	0	3	4	5	1	4	5
P3	0	0	3	6	3	5	6	3	2
P4	2	3	5	4	3	5	2	0	0
				14	10	18			

10

(a) Fill the remaining values in the 'Still Need' column for P2, P3, and P4. (3 points)

(b) Is the current state a safe or not? If safe, give a possible execution order. (3 points)

Yes:

P1 first: $0 \le 1$ $0 \le 1$ $2 \le 2$ Avail: 2 1 5 (R1 R2 R3)

then P4 then: $2 \le 2$ $0 \le 1$ $0 \le 5$ Avail: 6 4 10

then P2: $1 \le 6$ $4 \le 4$ $5 \le 10$ Avail: 9 8 15

then P3: $6 \le 9$ $3 \le 8$ $2 \le 15$ Avail: 15 11 20

P3 #last:

(c) If P2 requests (1, 1, 2), can the request be granted immediately? Why or why not? If yes, shows a possible execution order after the request is granted. (4 points)

Nope! ~~Ea~~ P2 current R_1 R_2 R_3 3 1 2 ⇒ No released!

Free: R_1 R_2 R_3

0 0 0

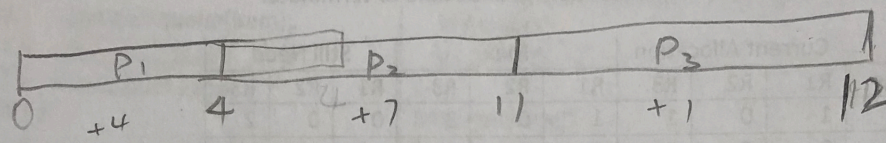
~~Ava~~ Ther other resources cannot be released!

11. [Scheduling] Suppose that the processes listed below (P1, P2, and P3) all arrive at the times indicated. Each process will run for the time listed. (15 points)

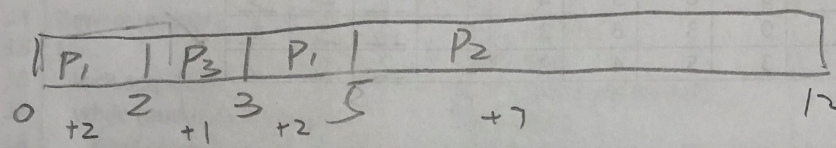
Name	Arrival Time	Burst Time
P1	0	4
P2	1	7
P3	2	1

(a) Draw timelines under three scheduling policies: First-Come First-Serve (FCFS), Shortest-Remaining-Time-First (SRTF), and Round-Robin (RR) with time quantum = 3. Assume that the context switch overhead is zero. Also, if there's a tie, lower index process should be scheduled first. (9 points)

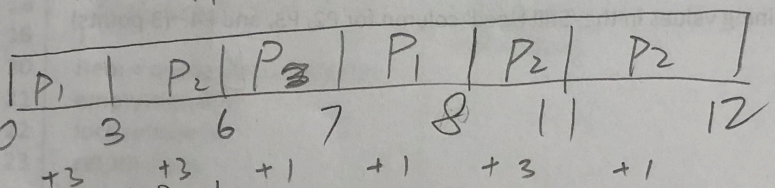
FCFS:



SRTF



RR



15

(b) Compute the average waiting time of each schedule (6 points)

FCFS: $(0 + (4-1) + (11-2))/3 = 4$ units

SRTF: $(11 + (5-1) + 0)/3 = \frac{5}{3} \approx 1.67$ units

RR: $(4 + (3+2-1) + (16-2))/3 = 4$ units