OS Example Questions

Part 1: True of False

- (a) DMA is a mechanism for allowing an I/O device to transfer data to and from memory without involving the CPU in the transfer.
- TRUE. *Direct memory access (DMA) is a feature of computer systems that allowscertain hardware subsystems to access main system memory (random-access memory), independent of the central processing unit (CPU).*
- (b) Memory mapped I/O determines how the pages of an I/O-bound process are mapped to page frames.
- FALSE
- (c) The root directory of a partition in a Unix system is named “/”.
- TRUE
- (d) There is only one MBR (master boot record) on a disk drive, but there could be several boot sectors.
- FALSE, In a single disk drive, you can have only one boot sector

Part 1: True of False

- (e) A context switch from one process to another can be accomplished without executing OS code in kernel mode.
- FALSE
- (f) An advantage of implementing threads in user space is that they don't incur the overhead of having the OS schedule their execution.
- TRUE
- (g) Deadlock can never occur if no process is allowed to hold a resource while requesting another resource.
- TRUE
- (h) In round robin scheduling, it is advantageous to give each I/O bound process a longer quantum than each CPU-bound process (since this has the effect of giving the I/O bound process a higher priority)
- FALSE

Part 1: True of False

- (i) For machines with 32-bit addresses (i.e. a 4GB address space), since 4GB physical memories are common and cheap, virtual memory is really no longer needed.
- FALSE
- (j) A TLB miss could occur even though the requested page was in memory.
- TRUE

Part II: Virtual Memory

- Suppose you had a computer that supported virtual memory and had 32-bit virtual addresses and 4KB (2^{12} byte) pages. If a process actually uses 1024 (2^{10}) pages of its virtual address space, how much space would be occupied by the page table for that process if a single-level page table was used? Assume each page table entry occupies 4 bytes.

There are $2^{32}/2^{12} = 2^{20}$ pages in the address space. Therefore, a single level page table would have 2^{20} entries, where each entry is 4 bytes. Therefore, the page table would require $4 * 2^{20}$ bytes = 4 MB. This is independent of how many pages are actually being used.

Part II: Virtual Memory

- Suppose, in the same machine as above, a two-level page table was used, such that the first level page table was four times the size of each second level page table. Do you have enough information to know exactly how much space is occupied by the two-level page table for that process? Explain.

No. How the used pages are distributed across the address space of the process will determine how many second-level page tables are needed.

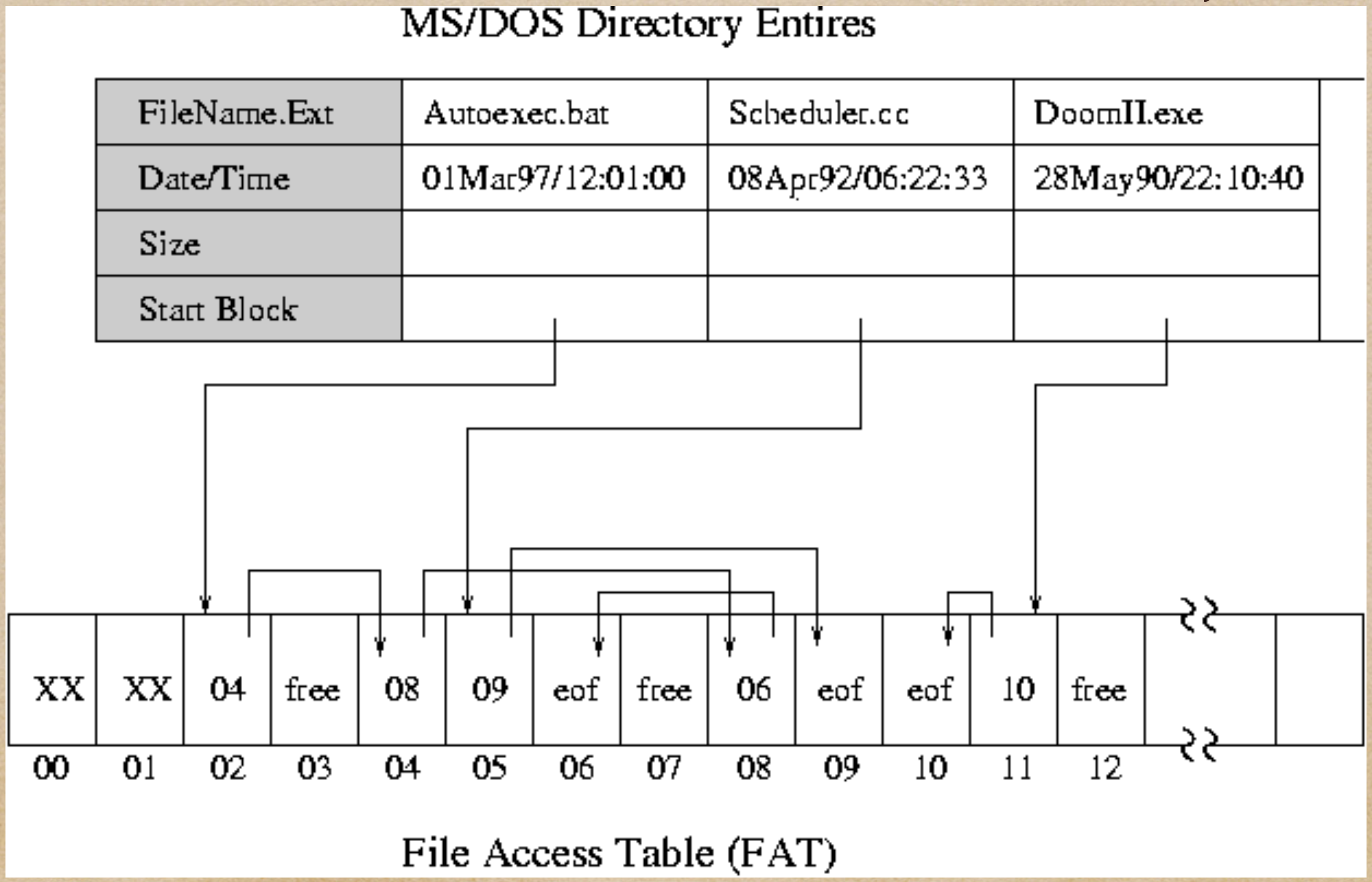
Part II: Virtual Memory

- compute the minimum amount of space that could possibly be required for the two-level page table for that process and compute the maximum amount of space that could possibly be occupied by the two-level page table for that process.

First, we need to compute the sizes of the first and second level page tables. Since there are 2^{20} pages and the first-level page table is four times the size of each second level page table, the first level page table must have 2^{11} entries and each second level page table must have 2^{9} entries (since their product must equal 2^{20}). Further, since each page table entry occupies 4 bytes (2^{2} bytes), the first level page table occupies $2^{11} * 2^{2} = 2^{13}$ bytes (8KB) and each second level table occupies $2^{9} * 2^{2} = 2^{11}$ bytes (2KB).

Part III: File

- Suppose a computer has a file system for a 128GB (2^{37} byte) disk, where each disk block is 8KB (2^{13} bytes). If the OS for this computer uses a FAT, what is the smallest amount of memory that could possibly be used for the FAT (assuming the entire FAT is in memory)? Explain.



Part III: File

- There has to be a FAT entry for each disk block. Since the disk is 2^{37} bytes and a disk block is 2^{13} bytes, the number of disk blocks (and thus the number of FAT entries) is $2^{37}/2^{13} = 2^{24}$. Since there are 2^{24} entries, a block number (disk address) requires a minimum of $\log(2^{24})$ bits = 3*8 bits = 3 bytes. In this case, the minimum amount of space occupied by the FAT is the number of entries (2^{24}) times the 3 bytes per entry, namely $2^{24} * 3$ = 16MB * 3 = 48MB.

Part III: File

- Suppose that on a different computer, the OS uses i-nodes and each disk block is 4KB (2^{12} bytes). Assume that an i-node contains 12 direct block numbers (disk addresses) and the block numbers for one indirect block, one double indirect block, and one triple indirect block. Assume also that a block number is 4 bytes. What is the largest possible file on that computer (assuming the disk is large enough).

Since a disk block is 4KB (2^{12} bytes) and a block number is 4 (2^2) bytes, there are 2^{10} = 1024 entries per indirect block. Therefore, the maximum number of blocks of a file that could be referenced by the i-node is $12 + 2^{10} + (2^{10})^2 + (2^{10})^3 = 12 + 2^{10} + 2^{20} + 2^{30}$. Thus, the maximum size of the file would be $(12+2^{10}+2^{20}+2^{30})*2^{12} = (12*2^{12})+2^{22}+2^{32}+2^{42} =$ 48KB + 4MB + 4GB + 4TB.

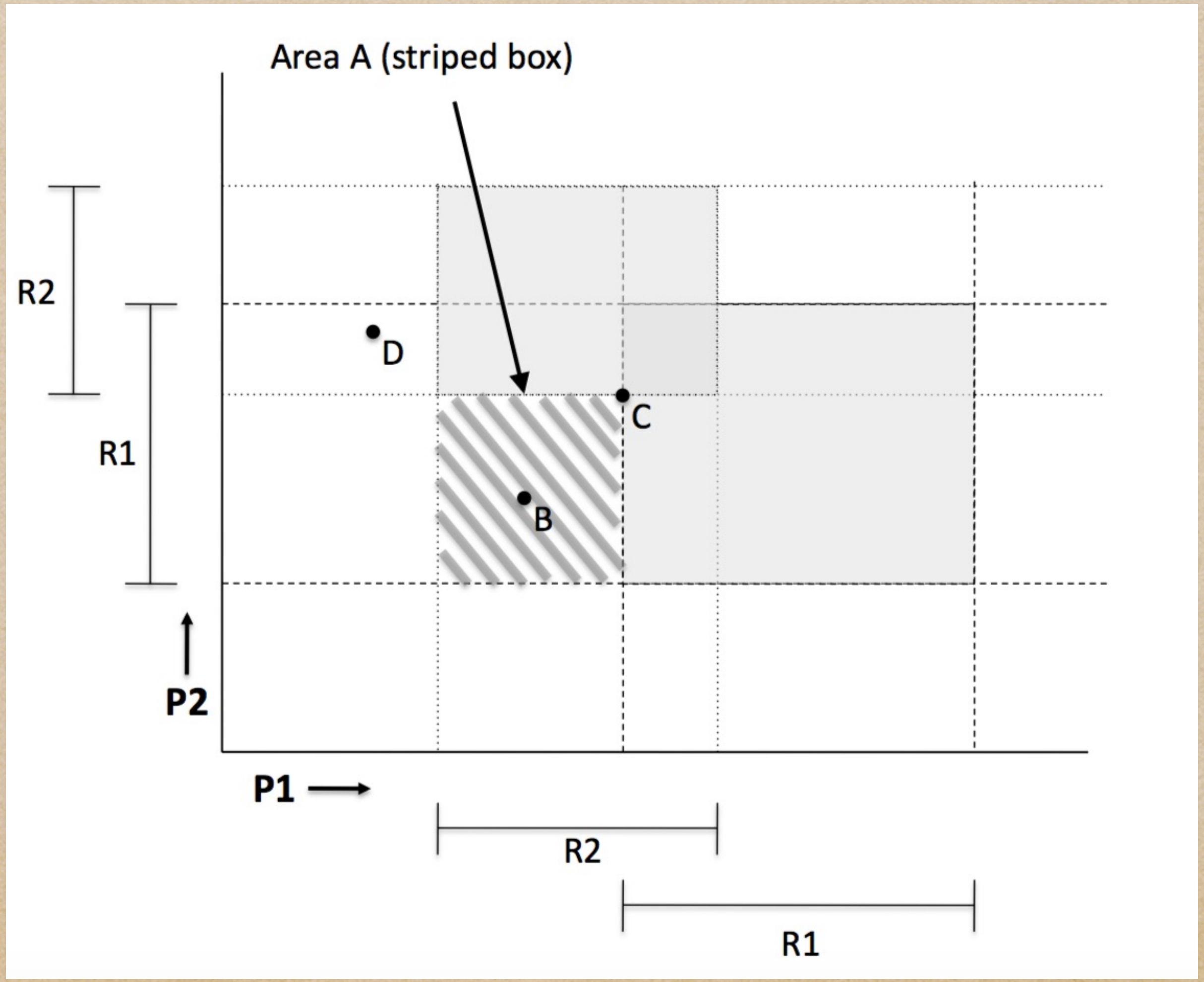
Part III: File

- On the computer from the last question, suppose you wanted to create a new file of the maximum size. How many free blocks on the disk would there need to be in order to create that file?

As discussed above, you would need $12 + 2^{10} + 2^{20} + 2^{30}$ blocks just to contain the data in the file itself. However, you would also need free blocks to use for your single indirect, double indirect, and triple indirect blocks. The total of those blocks is $1+(1+2^{10})+(1+2^{10}+2^{20}) =$ $3+(2*2^{10})+2^{20}$. Adding together the file blocks and the indirect blocks gives $(12+2^{10}+2^{20}+2^{30})+(3+(2*2^{10})+2^{20}) =$ $15+(3*2^{10})+(2*2^{20})+2^{30}$.

Part IV: Deadlock

- Consider the following resource trajectory graph for a system with two processes (P1 and P2) and two resources (R1 and R2).



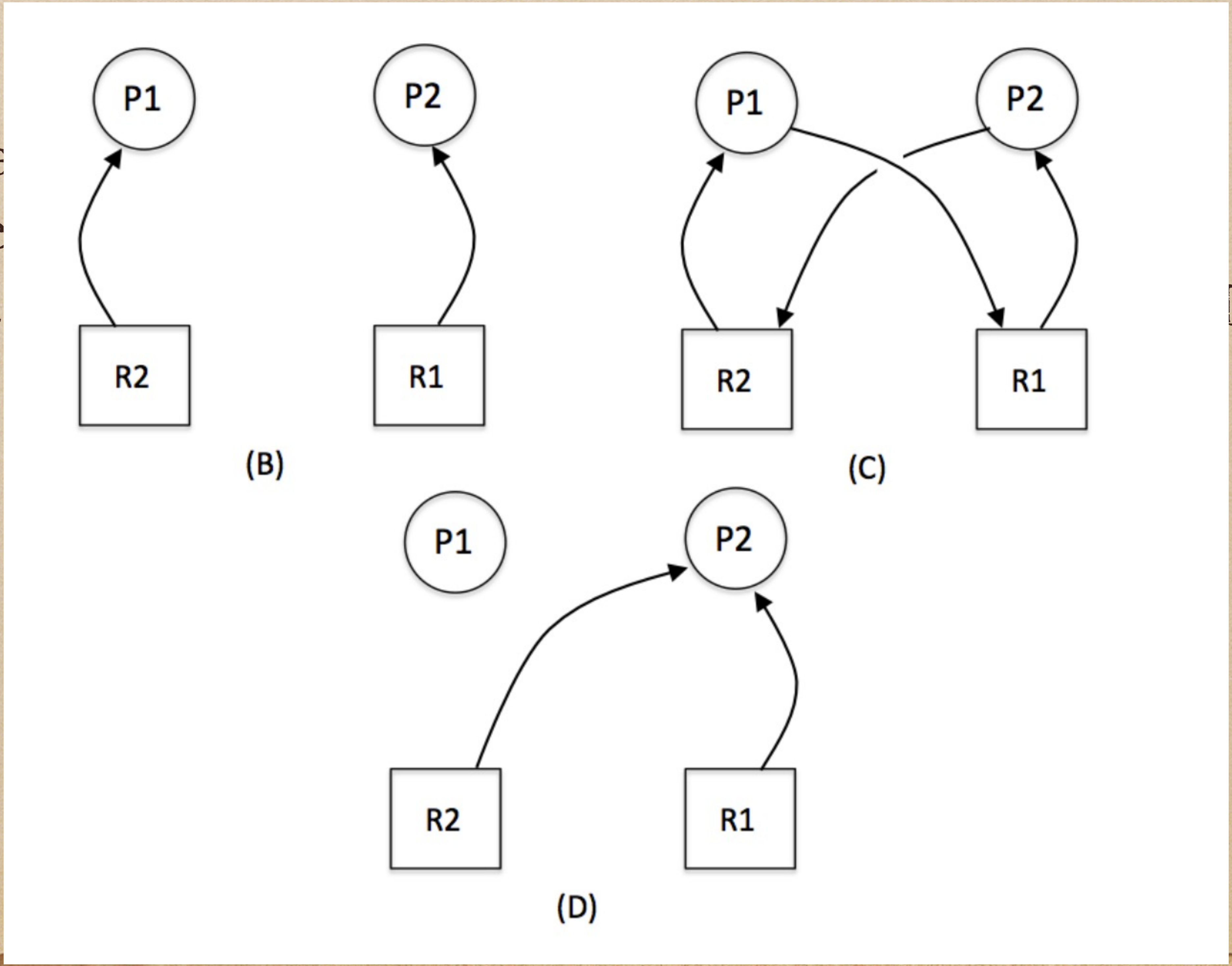
Part IV: Deadlock

- What does the area identified as Area A indicate?

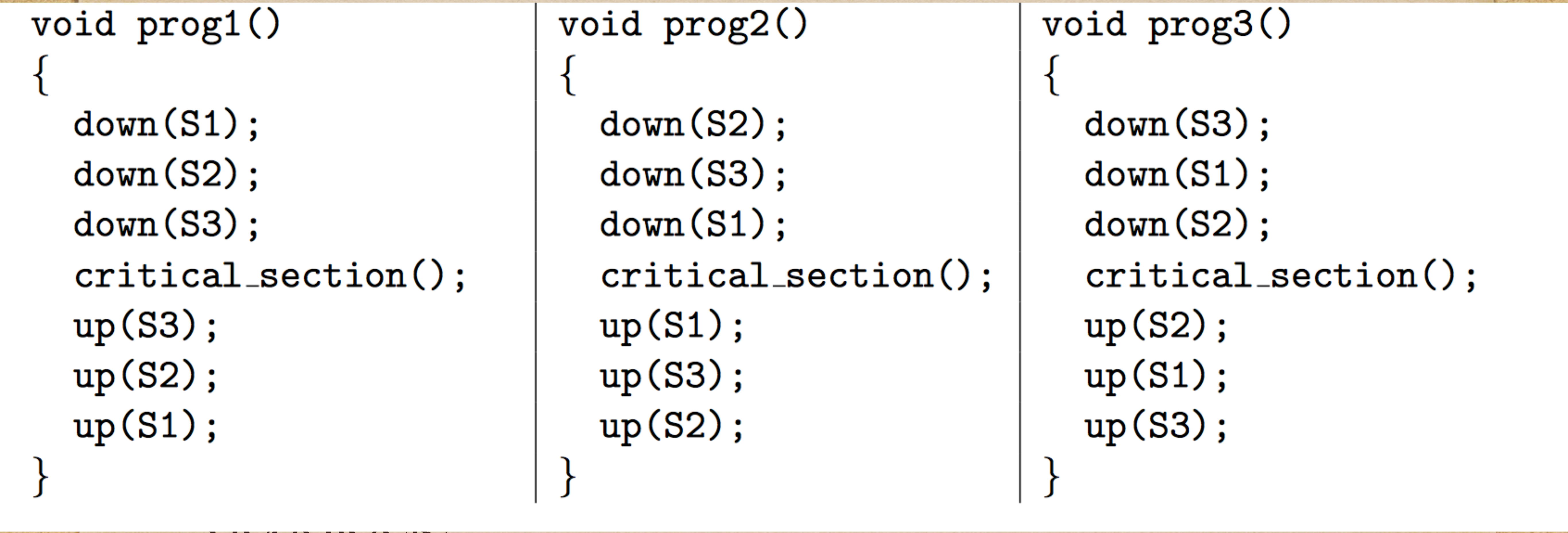
A region where the processes are not deadlocked, but will inevitably be deadlocked (when they hit point C)

Part IV: Deadlock

- Dra
  one
  eac D



Part IV: Deadlock



```
void prog1()
{
  down(S1);
  down(S2);
  down(S3);
  critical_section();
  up(S3);
  up(S2);
  up(S1);
}
```

```
void prog2()
{
  down(S2);
  down(S3);
  down(S1);
  critical_section();
  up(S1);
  up(S3);
  up(S2);
}
```

```
void prog3()
{
  down(S3);
  down(S1);
  down(S2);
  critical_section();
  up(S2);
  up(S1);
  up(S3);
}
```

deadlock?

Part IV: Deadlock

- Assume that each program has to declare, when it starts, which semaphores it will be using. In what way would the bankers algorithm, assuming the OS used it, prevent the deadlock in the scenario you gave in your previous answer?

Since each of the above programs will have declared that it will be using all three semaphores, once prog1 has succesfully executed down(S1), any success- ful down operation by the other programs would result in an unsafe state, therefore the OS would cause a down operation by prog2 or prog3 to block.

Part IV: Deadlock (banker)

- Consider the following snapshot of a system:

	Allocation	Max	Available
	ABCD	ABCD	ABCD
P0	0012	0012	1520
P1	1000	1750	
P2	1354	2356	
P3	0632	0652	
P4	0014	0656	

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix Need?

b. Is the system in a safe state?

c. If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?

Part IV: Deadlock (banker)

- a.

P0 0000;

P1 0750;

P2 1002;

P3 0020;

P4 0642;

	Allocation	Max	Available
	ABCD	ABCD	ABCD
P0	0012	0012	1520
P1	1000	1750	
P2	1354	2356	
P3	0632	0652	
P4	0014	0656	

- b.

Yes, P0 and P3 could get resources to run. After they finishing their jobs and release the resources, available is 1,11,6,4. Then others can run at the sequence < P0,P3,P2,P1,P4>. So it is at the safe state.

- c.

Yes. Available is 1520>0420; Now the resources is 1100; P0 can run and release the resources it holds. And they can run following the sequence <P0,P2,P3,P1,P4>. So it is also at the safe state.

Part V: I/O

- What is the advantage of the elevator algorithm over the shortest-seek-first algorithm?

In the presence of a stream of incoming block requests, the shortest seek first algorithm, which always chooses the block request to satisfy that requires the least movement of the head, could lead to starvation of requests for blocks that are far from the current head position.

Part VI: CPU Scheduling

- On some computer, the clock interrupt handler needs 2 msec (including context switch overhead) per clock tick to execute, and the clock runs at 75 Hz. What fraction of the CPU time is devoted to the clock?

The clock interrupt handler is run 75 times every second, which takes 75*2 = 150ms. This is 15% of the CPU time.

Part VI: CPU Scheduling

- Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

a. Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a non-preemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.

b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of the scheduling algorithms in part a?

d. Which of the schedules in part a results in the minimal average waiting time (over all processes)?

Part VI: CPU Scheduling

- a.

FCFS: P1-10-P2-1-P3-2-P4-1-P5-5

SJF: P2-1-P4-1-P3-2-P5-5-P1-10

Priority: P2-1-P5-5-P1-10-P3-2-P4-1

RR: P1-1-P2-1-P3-1-P4-1-P5-1-P1-1-P3-1-P5-1-P1-1-P5-1-P1-1-P5-1-P1-1-P5-1-P1-5

- b.

	P1	P2	P3	P4	P5
FCFS	10	11	13	14	19
SJF	19	1	4	2	9
Priority	16	1	18	19	6
RR	19	2	7	4	14

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

Part VII: Transaction Scheduler

- If we applied timestamp-Based Protocols on the following schedule:

	T0	*T1*	*T2*
Timestamp:	2	4	6
Operation:	R(A)	R(A)	R(A)
	W(A)		W(A)

If the execution sequence happens like:

T0’s R(A)
T0’s W(A)
T2’s R(A)
T2’s W(A)
T1’s R(A)
Is there any transaction need “Roll Back”? If yes, which one and why? If no, just say no.

Part VII: Transaction Scheduler

	T0	*T1*	*T2*
Timestamp:	2	4	6
Operation:	R(A)	R(A)	R(A)
	W(A)		W(A)

If the execution sequence happens like:

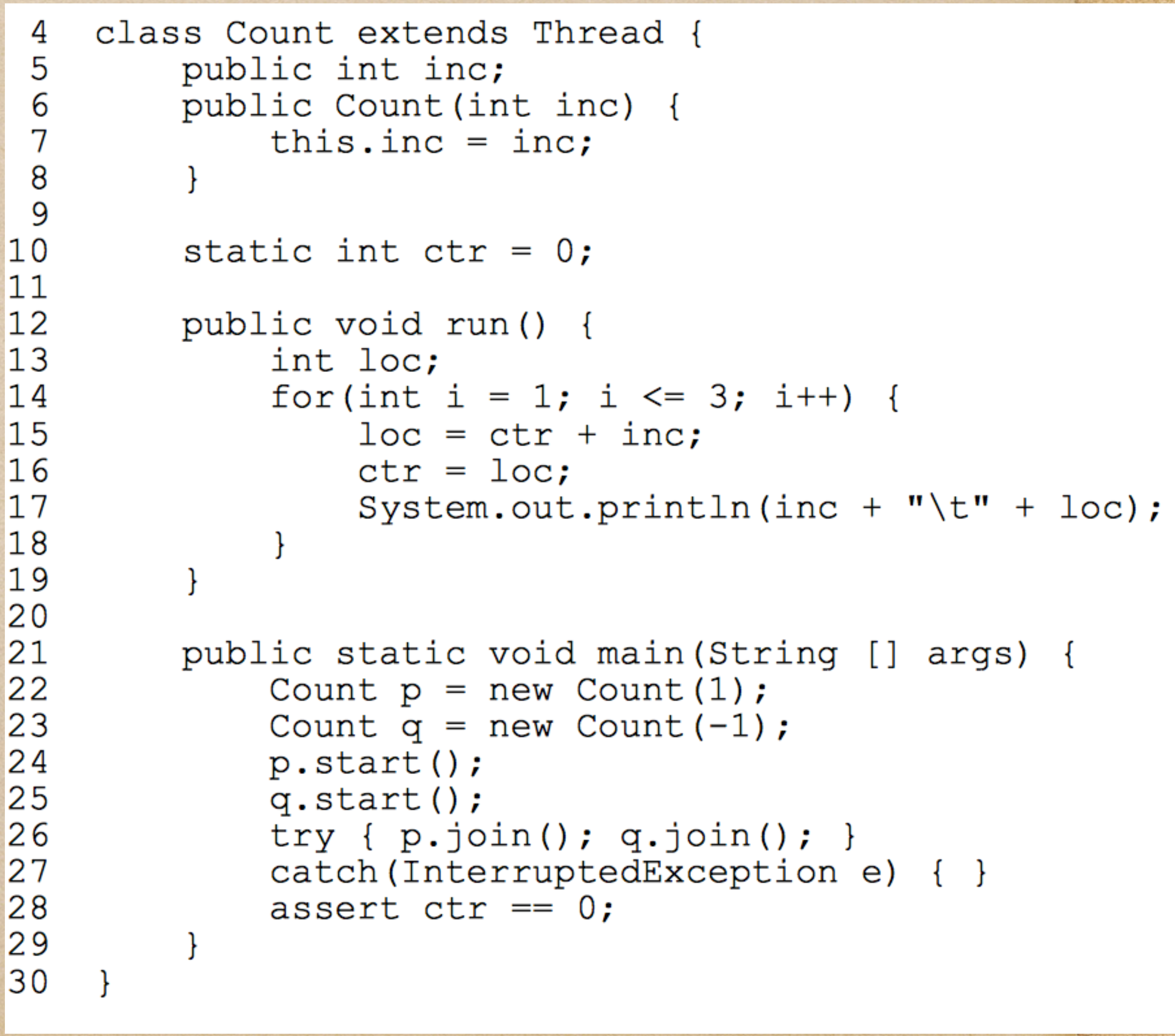
T0’s R(A)
T0’s W(A)
T2’s R(A)
T1’s R(A)
T2’s W(A)

If the execution sequence happens like:

T0’s R(A)
T1’s R(A)
T0’s W(A)
T2’s R(A)
T2’s W(A)

Part VIII: Racing

- Consider the java code
- what's the problem?
- How to address



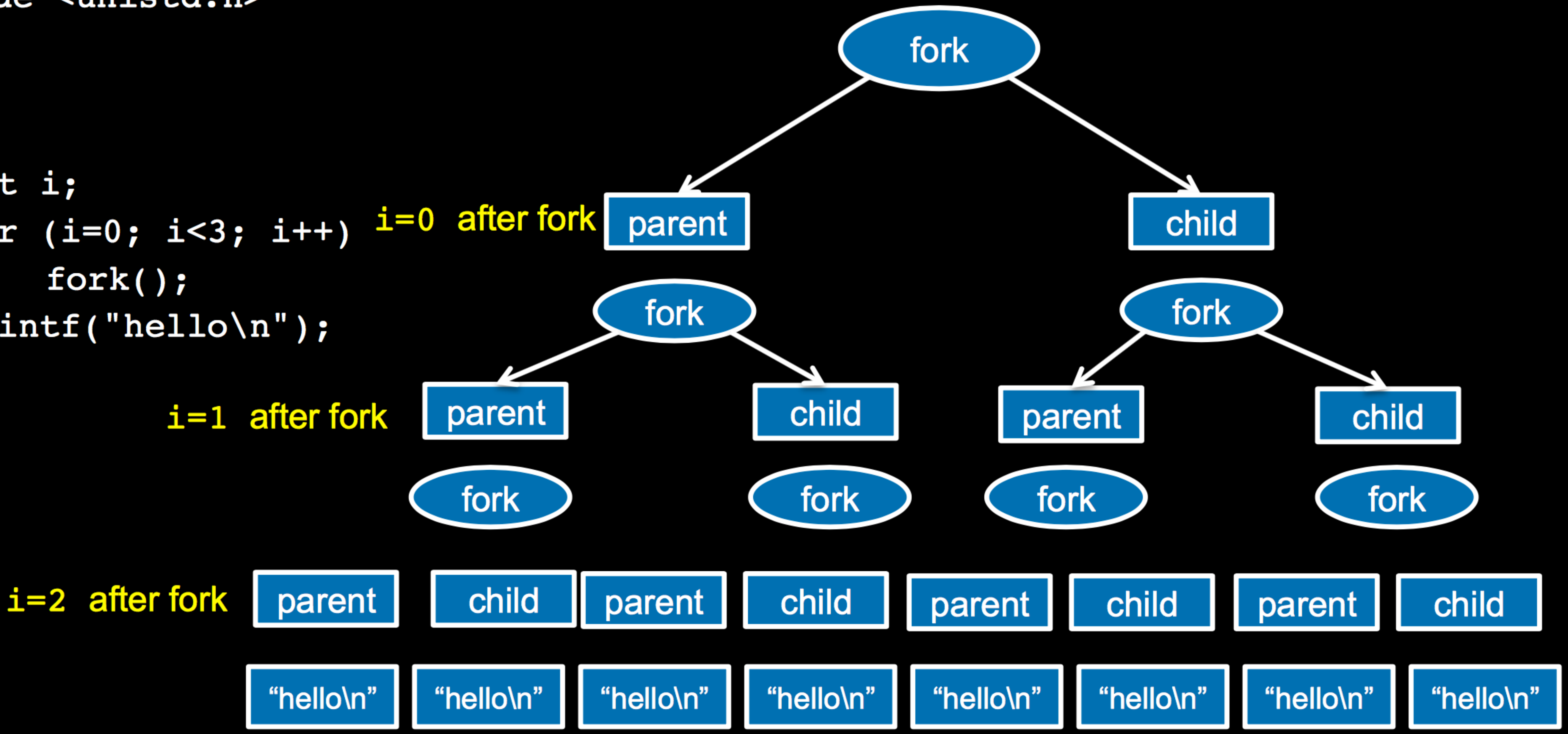
```
 4  class Count extends Thread {
 5      public int inc;
 6      public Count(int inc) {
 7          this.inc = inc;
 8      }
 9
10      static int ctr = 0;
11
12      public void run() {
13          int loc;
14          for(int i = 1; i <= 3; i++) {
15              loc = ctr + inc;
16              ctr = loc;
17              System.out.println(inc + "\t" + loc);
18          }
19      }
20
21      public static void main(String [] args) {
22          Count p = new Count(1);
23          Count q = new Count(-1);
24          p.start();
25          q.start();
26          try { p.join(); q.join(); }
27          catch(InterruptedException e) { }
28          assert ctr == 0;
29      }
30  }
```

Before or after line 10:
```
  static Semaphore s = new Semaphore(1)
```
Before line 15:
```
   try { s.acquire(); }
      catch(InterruptedException e) {}
```
After line 16:   s.release();

Part VIIII: Fork

```
#include <unistd.h>

main()
{
    int i;
    for (i=0; i<3; i++)
        fork();
    printf("hello\n");
}
```



Answer: 8

Part X Process Synchronization

Four processes R1, R2, W1, and W2 share a buffer space named B. R1 reads a number from the keyboard and saves it into B, so that the saved number is only consumed by W1, which prints the number to the screen. R2 reads a character from a mouse and also saves the character into B, so that only W2 can print the character to a printer. Please write the synchronization code for the four processes so that no race condition may arise among them. Define semaphores if necessary.

Part X Process Synchronization

```
Semaphore S=1, S1=0, S2=0;
Buffer B;
R1()
{    int n;
     while (1){
          n=getKeyboardInput();
          wait(S);
          B=n;
          Signal(S1);
     }
}
W1()
{    int n;
     while(1){
          wait(S1);
          n=B;
          signal(S);
          print n;
     }
}
```

```
R2()
{    char c;
     while (1){

     c=getMouseInput
();
          wait(S);
          B=c;
          signal(S2);
     }
}
W2()
{
     char c;
     while(1){
          wait(S2);
          c=B;
          signal(S);
          print c;
     }
}
```

Part XI Page Replacement

- A page-replacement algorithm should minimize the number of page faults. We can do this minimization by distributing heavily used pages evenly over all of memory, rather than having them compete for a small number of page frames. We can associate with each page frame a counter of the number of pages that are associated with that frame. Then, to replace a page, we search for the page frame with the smallest counter.

a. Define a page-replacement algorithm using this basic idea. Specifically address the problems of (1) what the initial value of the counters is, (2) when counters are increased, (3) when counters are decreased, and (4) how the page to be replaced is selected.

Part XI Page Replacement

- a.

(1) Initial value of the counters -- 0

(2) Counters are increased -- whenever a new page is associated with that frame

(3) Counters are decreased -- whenever one of the pages associated with that frame is no longer required

(4) How the page to be replaced is selected -- find a frame with the smallest counter. Use FIFO for breaking ties.

Part XI Page Replacement

b. How many page faults occur for your algorithm for the following reference string, for four page frames?

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.

c. What is the minimum number of page faults for an optimal page-replacement strategy for the reference string in part b with four page frames?

Part XI Page Replacement

- b. The number of page faults is 13.

```
1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2
1  1  1  1  5        5  5  5  8        8         8  8       2
   2  2  2  2        1  1  1  1        9         9  9       9
      3  3  3        3  6  6  6        6         6  5       5
         4  4        4  4  7  7        7         7  4       4
```

- c. The number of page faults is 11 for optimal page-replacement strategy.

```
1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2
1  1  1  1  1           6  7  7           7           4      4
   2  2  2  5           5  5  5           5           5      5
      3  3  3           3  3  8           8           8      2
         4  4           4  4  4           9           9      9
```

Part XII Demand Paging

- Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 milliseconds.
- Addresses are translated through a page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two accesses.
- To improve this time, we have added an associative memory that reduces access time to one memory reference.
- Assume that 80 percent of the accesses are in the associative memory and that, of the remaining, 10 percent (or 2 percent of the total) cause page faults. What is the effective memory access time?

Part XII Demand Paging

- EAT has 3 parts :
- The first part: "to improve time, we have added an associative memory that reduces access time to one memory reference, if the page-table entry is in the associative memory." In this part, only 1 memory access is needed since it is in associative memory. Time1=80%*1us.
- The second part: In memory but not in associative memory. 2 memory accesses are needed which is told in the problem description: one for associative memory and one for normal memory since it is not found in associative memory. The amount of this part is 90% in 20% since the problem description told us "10 percent (or 2 percent of the total) cause page faults". Time2=18%*2us.
- The third part: Page faults. Paging disk access is needed. Time3=2% * (1us+20ms).
- EAT=(80%*1us)+(18%*2us)+(2%*20002us)=401.2us

Part XIII Disk Allocation

- Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For each of the three allocation strategies (contiguous, linked, and indexed), answer these questions:

a. How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation, assume that a file is always less than 512 blocks long.)

b. If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk?

Part XIII Disk Allocation

- Let z to be the starting file address.

- contiguous:

(a)

Divide the logical address by 512 with x and y, y is the remainder.

x+z is the block number, y is the offset

(b)

Directly access the block 4. 1 block blocks must be reed.

Part XIII Disk Allocation

- linked:

(a)

Divide the logical physical address by 511 with x and y, y is the remainder.

Search the blocks down x+1, and the y+1 is the displacement into the last physical block.

(b)

Search for the list. 4 blocks must be reed.

Part XIII Disk Allocation

- indexed:

(a)

Divide the logical address by 512 with x and y, y is the remainder.

Physical block address is contained in the index block at location x. y is the offset of the block.

(b)

2 blocks must be reed. One for index block and one for the block needed.

Part XV Disk Scheduling

- Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999.

The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk-scheduling algorithms?

a. FCFS

b. SSTF

c. SCAN

d. LOOK

e. C-LOOK

f. C-SCAN

Part XV Disk Scheduling

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

- a. FCFS:

143, 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

TOTAL = 7081

- b. SSTF:

143, 130, 86, 913, 948, 1022, 1470, 1509, 1750, 1774

TOTAL = 1745

- c. SCAN

143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 130, 86

TOTAL = 2*4999-143-86 = 9769

Part XV Disk Scheduling

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

- d. LOOK

143, 913, 948, 1022, 1470, 1509, 1750, 1774, 130, 86

TOTAL = 1774*2-143-86=3319

- e. C-LOOK

143, 913, 948, 1022, 1470, 1509, 1750, 1774, 86, 130.

TOTAL = 1774*2-143-86+ (130-86) = 3363