University of Waterloo
Midterm Examination
Term: Winter Year: 2010

Solution

begin solution

Grade breakdown: Winter 2010

Question	1	2	3	4	5	6	7	8	Total
Average	7.5	21.2	12.6	17.8	9.0	9.0	7.2	22.6	107.0
Max	12.0	24.0	20.0	24.0	16.0	15.0	9.0	25.0	137.0
Out of	12.0	24.0	20.0	24.0	16.0	15.0	9.0	25.0	145.0
Average %	62.4	88.5	62.8	74.3	56.5	60.3	79.9	90.2	73.8

90-100	9
80-89	34
70-79	50
60-69	23
50-59	12
40-49	4
30-39	1
20-29	0
10-19	1

end solution

Problem 1 (12 marks)

a. **(4 mark(s))** There are only two processes in the system P_A and P_B containing one thread each. Assume that P_A is running and P_B is on the ready to run queue. Describe the steps required to save and restore or manipulate the processes context (including MMU) if P_A performs a **non-blocking** system call and it has not used up all of its quantum. Be sure to explain where the context is saved to and restored from. Provide a general description, you don't need to give specific register names.

——————begin solution——————

- On the trap into the kernel all of the processor's registers are saved onto P_A's trap frame (onto the kernel stack for P_A). This is P_A's user-mode context.
- After executing the system call P_A's user-mode context is retreived from the trap frame in it's kernel stack and restored to the processor registers just prior to returning from the exception to continue running in user mode.

——————end solution——————

b. **(8 mark(s))** There are only two processes in the system P_C and P_D containing one thread each. Assume that P_C is running and P_D is on the ready to run queue. Describe the steps required to save and restore or manipulate the processes context (including MMU) if P_C performs a **blocking** system call and it has not used up all of its quantum. Be sure to explain where the context is saved to and restored from. Provide a general description, you don't need to give specific register names.

——————begin solution——————

- On the trap into the kernel all of the processor's registers are saved onto P_C's trap frame (onto the kernel stack for P_C). This is P_C's user-mode context.
- When P_C blocks, we save it's kernel-mode context onto the it's kernel stack.
- We then restore P_D's context from it's kernel stack. This is it's kernel-mode context.
- Flush/Invalidate the contents of the TLB.
- Then P_D's user-mode context is retreived from the trap frame in it's kernel stack and restored to the processor registers just prior to returning from the exception to continue running in user mode.

——————end solution——————

Problem 2 (24 marks)
For this question all addresses, virtual page numbers and physical frame numbers are represented in hexidecimal. Consider a machine with *44-bit* virtual addresses, *48-bit* physical addresses, and a page size of 1 MB. During a program execution the TLB contains the following valid entries (in hexidecimal).

Virtual Page Num	Physical Frame Num	Valid	Read-Only / Can't Make Dirty
0x 0	0x 171	0	0
0x 1E	0x 172	1	1
0x 1EF	0x 173	1	1
0x 1EF1	0x 1EF	1	0
0x 1EF17	0x EF1	0	0
0x 1EF170	0x 8132CD	1	0
0x 1EF171	0x 132CD	1	1
0x 1EF172	0x 88132CD	1	0

Examine the following set of instructions and if possible, translate the addresses. If the translation is not possible or an exception would be raised, indicate which exception and why the exception is raised. Show and explain how you derived your answer. Express, in hexidecimal, the required address using **ALL 44-bits for virtual addresses and 48-bits for physical addresses** (i.e., including leading zeros).

————————begin solution————————

1 MB page size = 2^{20} so 5 hex digits for offset.
44-20 = 24 bits (6 hex digits for virtual page number.

————————end solution————————

a. **(4 mark(s))** A load occurs from virtual address = 0x 000 01EF AC00.
If a translation occurs provide the physical address.

————————begin solution————————

```
Offset = F AC00
Virtual Page = 1E
Valid bit = 1
Translation occurs
Frame number = 172
Phyical address = 0x 0000 172F AC00
```

————————end solution————————

b. **(4 mark(s))** A load occurs from virtual address = 0x 1EF 1729 52CD.
If a translation occurs provide the physical address.

————————begin solution————————

```
Offset = 9 52CD
Virtual Page = 1EF172
Entry is found in the TLB.
Valid bit = 1
Translation occurs
Frame number = 88132CD
Phyical address = 0x 8813 2CD9 52CD
```

———————end solution———————

c. **(4 mark(s))** A store occurs to virtual address = 0x 1EF 1729 AC00.
If a translation occurs provide the physical address.

———————begin solution———————

```
Offset = 9 AC00
Virtual Page = 1EF172
Entry is found in the TLB.
Valid bit = 1
Translation occurs
Frame number = 88132CD
Phyical address = 0x 8813 2CD9 AC00
```

———————end solution———————

d. **(4 mark(s))** A store occurs to virtual address = 0x 1EF 1709 AC00.
If a translation occurs provide the physical address.

———————begin solution———————

```
Offset = 9 AC00
Virtual Page = 1EF170
Entry is found in the TLB.
Valid bit = 1
Read only bit = 0
Translation occurs.
Phyical address = 0x 0813 2CD9 AC00
```

———————end solution———————

e. **(4 mark(s))** A load occurs from virtual address = 0x 1EF 1739 AC00.
If a translation occurs provide the physical address.

———————begin solution———————

```
Offset = 9 AC00
Virtual Page = 1EF173
No entry is found in the TLB.
Translation fails.
TLB miss exception is raised.
```

———————end solution———————

f. **(4 mark(s))** Can a store occur at physical address 0x 0000 1EF1 32CD? Explain your answer. If it can occur, what is the corresponding virtual address?

———————begin solution———————

```
Offset = 1 32CD
Physical Page = 1EF
There is an entry for that physical page in the TLB.
Valid = 1
Read only = 0
So translation could have occurred.
The virtual page number is 1EF1

So yes, the store could occur and the virtual address is
0x 001 EF11 32CD
```

end solution

Problem 3 (20 marks)

a. **(12 mark(s))** We would like multiple threads to be able to call `add` and `sub` (shown below), simultaneously. Add code that uses **locks** (as defined in OS/161) to ensure that all of the code below will execute correctly and so that maximum parallelism is ensured. Assume that these are the only functions that can alter the array and that there is no need to do any error checking.

```
#define MAX      (100)
volatile int array[MAX];









void init() /* Only called by one thread before using other functions */
{
        int i;


        for (i=0; i<MAX; i++) {


            array[i] = 0;


        }


}

void add(int index, int value)
{



        array[index] = array[index] + value;



}

void subtract(int index, int value)
{



        array[index] = array[index] - value;



}
```

begin solution

```
#define MAX      (100)
volatile int array[MAX];

/* **** ADD **** */
struct lock *locks[MAX];

void init() /* Only called by one thread before using other functions */
{
        int i;


        for (i=0; i<MAX; i++) {

            array[i] = 0;

            /* **** ADD **** */
            locks[i] = create_lock("name");
        }
}

void add(int index, int value)
{
    /* **** ADD **** */
    lock_acquire(locks[index]);


        array[index] = array[index] + value;

    /* **** ADD **** */
    lock_release(locks[index]);
}

void subtract(int index, int value)
{
    /* **** ADD **** */
    lock_acquire(locks[index]);

        array[index] = array[index] - value;

    /* **** ADD **** */
    lock_release(locks[index]);
}
```

end solution

b. **(8 mark(s))** Now you would like to add the `sum` function shown below to the library of functions on the previous page. This function will be called relatively infrequently (e.g., once a month) and **IT WILL ONLY EVER BE CALLED BY ONE THREAD**.

Assuming that multiple threads may be executing `add` and `sub` concurrently with the thread that calls `sum`, is synchronization required for the function `sum` if it is only called by a single thread? Explain why or why not. If synchronization is required, add it to the code below so that it integrates with your solution on the previous page (i.e., be sure to do the first part of this question first).

```
int sum()
{
        int i;
        int total = 0;









        for (i=0; i<MAX; i++) {




                total = total + array[i];




        }









        return total;




}
```

begin solution

```
/* One possibility */
int sum()
{
        int i;
        int total = 0;


        /* **** ADD **** */
        for (i=0; i<MAX; i++) {
           lock_acquire(locks[i]);
        }

        for (i=0; i<MAX; i++) {
                total = total + array[i];
        }

        /* **** ADD **** */
        for (i=0; i<MAX; i++) {
           lock_release(locks[i]);
        }

        return total;
}

/* Another possibility */
int sum()
{
        int i;
        int total = 0;

        for (i=0; i<MAX; i++) {
                /* **** ADD **** */
                lock_acquire(locks[i]);
                total = total + array[i];
        }

        /* **** ADD **** */
        for (i=0; i<MAX; i++) {
           lock_release(locks[i]);
        }

        return total;
}

/* And Another possibility */
int sum()
{
        int i;
        int total = 0;

        /* **** ADD **** */
```

```
        for (i=0; i<MAX; i++) {
                lock_acquire(locks[i]);
        }

        for (i=0; i<MAX; i++) {
                total = total + array[i];

                /* **** ADD **** */
                lock_release(locks[i]);
        }

        return total;
}
```

- Does need to be synchronized
- Other threads can change array while sum is being computed
- Best can do is to compute the sum of a snapshot. Snapshot is valid at the point that the final lock is grabbed.
- May note that the actual total could change between the time the first lock is released (e.g., locks[0] and the total is returned).

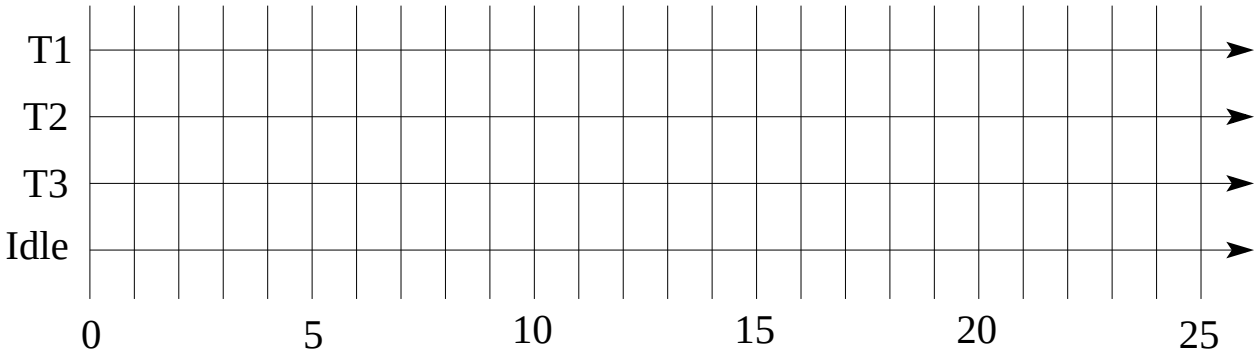
————————end solution————————

Problem 4 (24 marks)
Three threads are in the ready to run queue, in the order, T_1, T_2, and T_3.

- Thread T_1 runs a loop that executes 4 times. During each loop it runs for 1 unit of time and then makes a system call that blocks for 4 units of time. After looping the program exits.
- Threads T_2, and T_3 run a loop that executes 2 times. During each loop they run for 3 units of time and then call `thread_yield`. After looping the program exits.

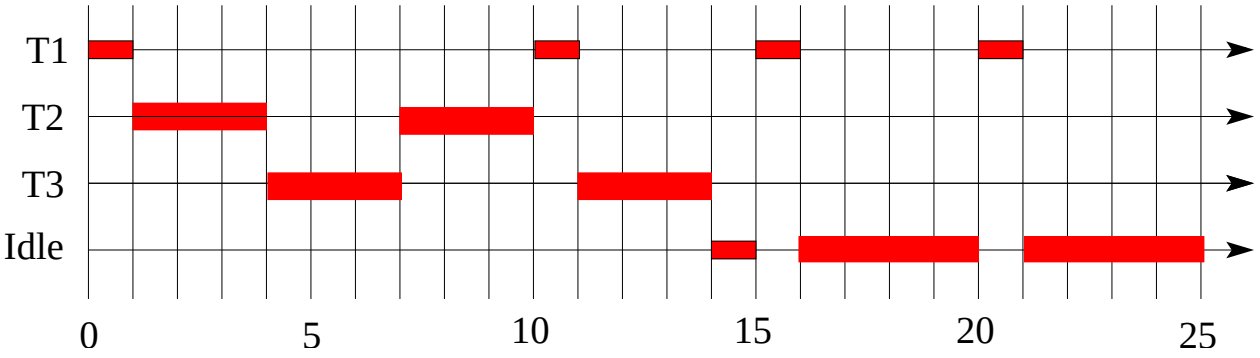
On the timeline below show when each thread runs and when the processor is idle by shading the appropriate thread or idle line. **If two events happen at the same time, assume that the higher numbered thread goes first (i.e., its event occurs first).**

a. **(10 mark(s))** Assume a non preemptive FIFO scheduler and the starting point is as described at the top of this question.



begin solution

NOTE: There was some confusion about the tie-breaking so
we need to be careful marking this question.
1 mark per line = 10 marks.



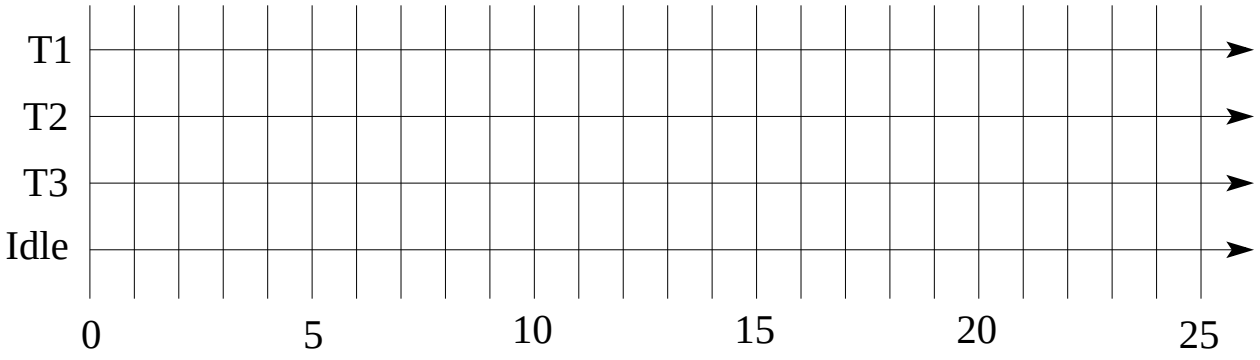
One way is to track the queue at each point in time
(below to simplify, we are running the thread at the front of the queue)

```
Time  Queue
t0    T1, T2, T3
t1    T2, T3        (T1 is blocked)
t4    T3, T2        (T1 is blocked)
t5    T3, T2, T1    (T1 is unblocked)
t7    T2, T1, T3    (T2 is finished)
t10   T1, T3
t11   T3            (T1 is blocked)
```

```
t14   Idle          (T3 is finished)
t15   T1            (T1 is unblocked)
t16   Idle          (T1 is blocked)
t20   T1            (T1 is unblocked)
t21   Idle          (T1 is blocked)
t25   Idle          (T1 is unblocked)
```

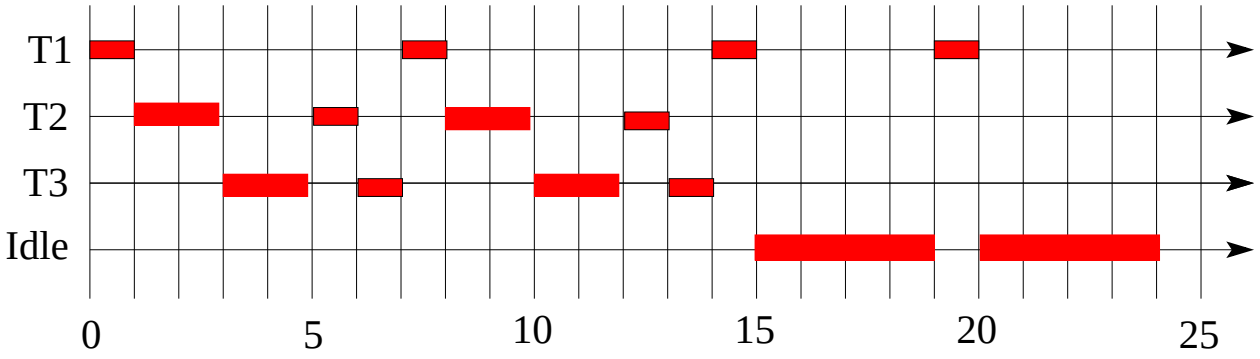
end solution

b. **(14 mark(s))** Assume a preemptive round-robin scheduler with a quantum of 2 and that the running time for the thread is reset to zero every time the thread executes (i.e., unused quantums do **not** carry over to the next time the thread runs). Use the starting point as described at the top of the question.



begin solution

```
THIS SOLUTION NEEDS TO BE CHECKED / REDONE
1 mark per line = 14 marks.
```



end solution

Problem 5 (16 marks)

a. **(2 mark(s))** After a thread has called `thread_yield` what steps are required for it to run again?

——————begin solution——————

It gets put into the ready to run queue.
It gets scheduled to run and gets dispatched.

——————end solution——————

b. **(2 mark(s))** Under what circumstances might an executing thread call `thread_yield` and yet be the next thread to run.

——————begin solution——————

When it is the only thread in the system.
When all other threads are blocked.

——————end solution——————

c. **(2 mark(s))** After a thread has called `thread_sleep` what steps are required for it to run again?

——————begin solution——————

Another thread must wake it up using thread wakeup.
It gets added to the ready to run queue.
The scheduler picks it to run and it is dispatched.

——————end solution——————

d. **(6 mark(s))** Describe one pro (advantage) and two cons (disadvantages) of disabling interrupts as a means for implementing synchronization.

——————begin solution——————

This and next question were generally poorly done. - reiterating the question as a statement - irrelevant facts about synchronization do not answer the question

pro: does not require any hardware specific synchronization instructions

plus two of these three:

con: will not enforce mutual exclusion on multiprocessors
con: ignoring timer interrupts has side effects (i.e., can cause problems)
con: prevents all preemption, not just preemption that would threaten the critical section

——————end solution——————

e. **(4 mark(s))** Describe one pro (advantage) and one con (disadvantage) of using an atomic instruction like Test-and-Set as a means for implementing synchronization.

——————begin solution——————

pro: it works on multiprocessor systems
con: it can waste cpu resources essentially spinning for a long time

——————end solution——————

Problem 6 (15 marks)
WRONG ANSWERS WILL RECEIVE NEGATIVE MARKS, SO YOU MAY NOT WANT TO GUESSS IF YOU ARE NOT SURE OF THE ANSWER

a. **(3 mark(s))** In OS/161 the only instruction that can be executed in user-mode that can cause the processor to switch to priveledged (or kernel) mode is the `syscall` instruction. True or False?

———————begin solution———————

FALSE.

———————end solution———————

b. **(3 mark(s))** The MIPS uses a software loaded TLB. True or False?

———————begin solution———————

TRUE.

———————end solution———————

c. **(3 mark(s))** In OS/161 interrupts can be disabled while in user-mode. True or False?

———————begin solution———————

FALSE

———————end solution———————

d. **(3 mark(s))** In OS/161 the number of bytes in the virtual address space of a program is always equal to the number bytes in the executable file True or False?

———————begin solution———————

FALSE

———————end solution———————

e. **(3 mark(s))**

The original implementation of semphores provided by OS/161 ensures that starvation can not occur. True or False?

———————begin solution———————

FALSE

———————end solution———————

Problem 7 (9 marks)
Study the code below and answer the question that follows.

```
/* SHOULD ALWAYS BE BETWEEN 0 AND 100. INCLUDING 0 AND 100 */
volatile int count = 0;

adjust_count()
{
    int spl;
    int doreset = FALSE;

    if (count == 100) {
       doreset = TRUE;
    }

    spl = splhigh();

    if (doreset == TRUE) {
        count = 0;
    } else {
        count++;
    }

    splx(spl);
}
```

a. **(9 mark(s))** If multiple threads are executing and calling `adjust_count`, is the value for the variable `count` guaranteed to be between 0 and 100 (including 0 and 100)? Explain your answer. Use examples if it helps to clarify your answer.

begin solution

The variable count can be modified by multiple threads. Because its value can change while being tested it needs to be inside the critical section.

If multiple threads check the count at the same time, and each finds the count equal to 99 each will increment the count and it will go beyond 100.

Common mistakes were thinking that the local stack variable `doreset` is shared between threads and comming up with an explanation where doreset gets read by one thread and changed by another thread (this is incorrect).

end solution

Problem 8 (25 marks)

This question uses the following notation (as used in the course notes) to describe resource allocation in a computer system :

- D_i: demand vector for process P_i
- A_i: current allocation vector for process P_i
- U: unallocated (available) resource vector

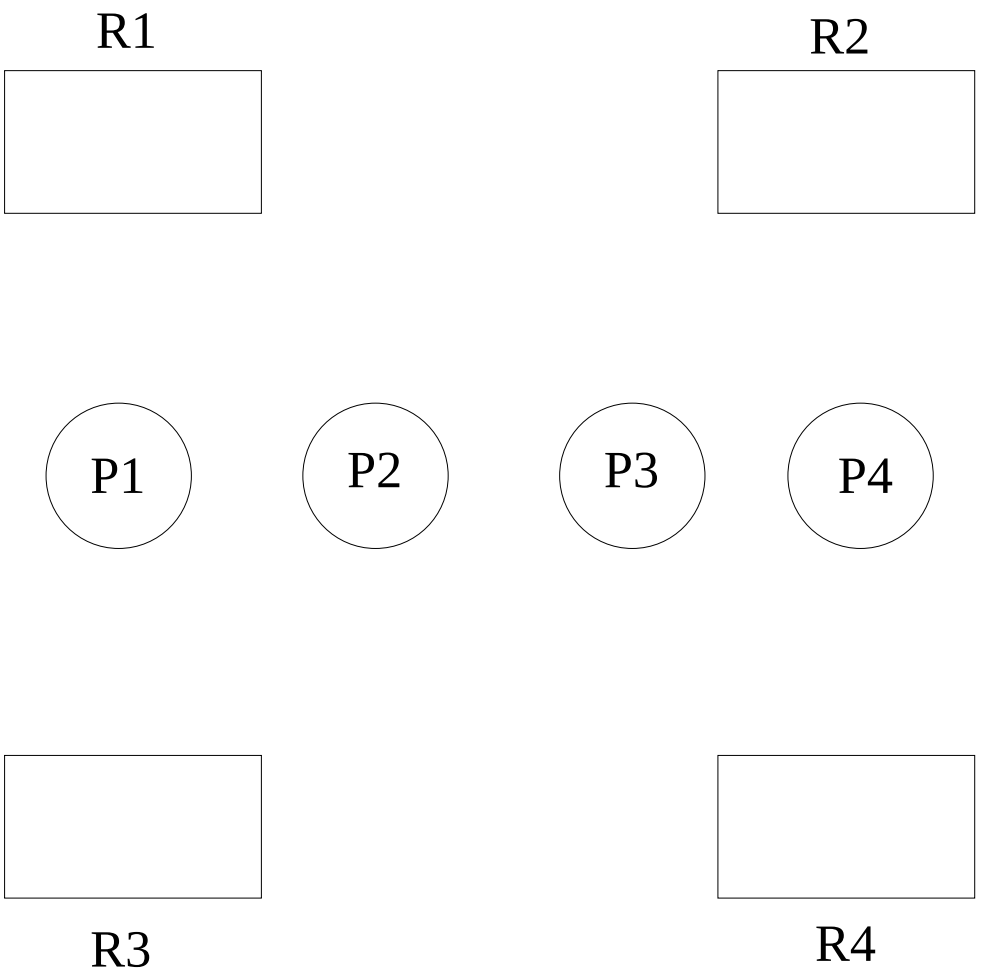
Given the scenario below, fill in the details for the resource allocation graph, **indicate if the system is deadlocked, and justify your answer (in one sentence).**

PLEASE USE SOLID LINES WITH ARROWS FOR ALLOCATION EDGES AND DASHED OR DOTTED LINES WITH ARROWS FOR REQUEST EDGES.

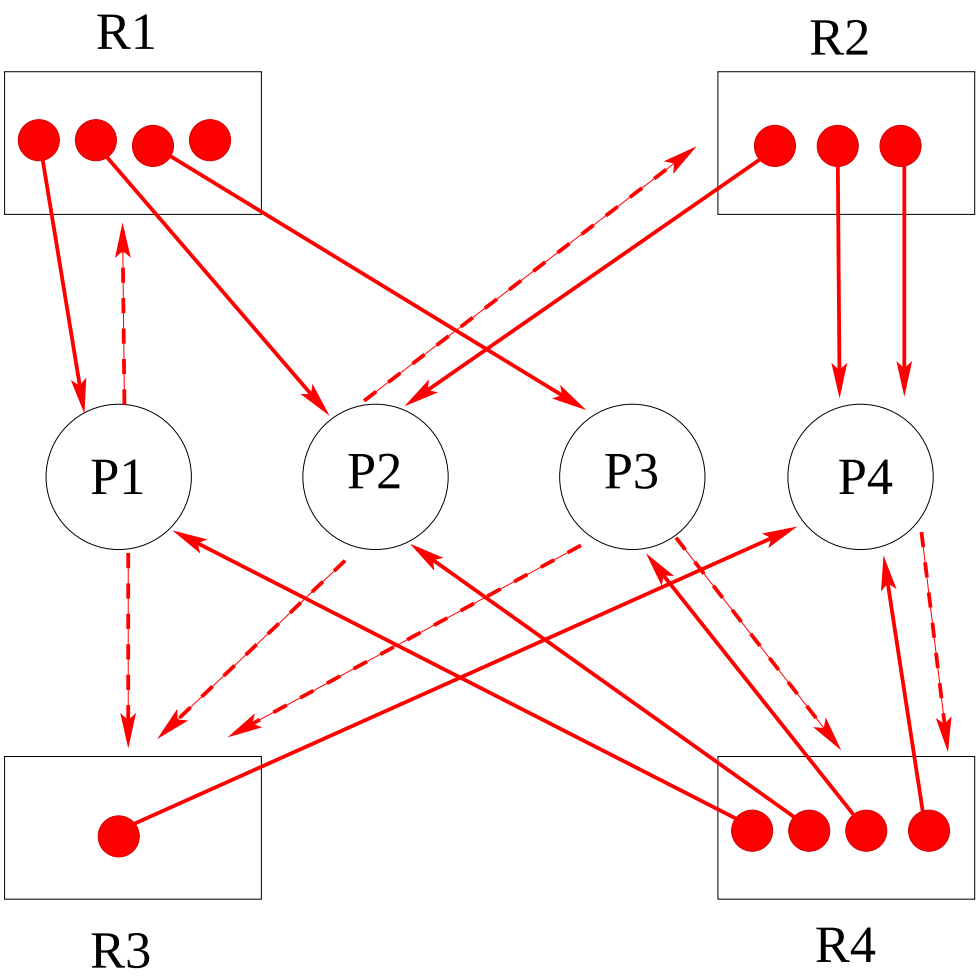
$U = (1, 0, 0, 0)$

$D_1 = (1, 0, 1, 0)$, $D_2 = (0, 1, 1, 0)$, $D_3 = (0, 0, 1, 1)$, $D_4 = (0, 0, 0, 1)$

$A_1 = (1, 0, 0, 1)$, $A_2 = (1, 1, 0, 1)$, $A_3 = (1, 0, 0, 1)$, $A_4 = (0, 2, 1, 1)$.



begin solution



The system IS DEADLOCKED.

The only resource available is one R_1 and

Each of P_1, P_2, P_3 and P_4 need an R_3 or an R_4 and there aren't any of those available so there is a deadlock.

Or the only unallocated resource is 1 R_1 and giving it to ANY of the P_i's doesn't help any of them to finish, so there is deadlock.

end solution