
Important Notes:

1. You have 150 minutes for answering the following questions. There are 8 questions in all. The points for each question are indicated in bold at the end of the question.
 2. The estimated time for each question is shown at the beginning of each question. Be concise and accurate in your answers. If you don't know the answer to a question, the best strategy is to move on, and return to the question later.
 3. Please answer the questions in the space provided below each question. If you need additional space, use the blank page at the end. However, you can use your own paper for rough work if necessary.
-

Q1 {10 minutes} The local laundromat has just entered the computer age. As each customer enters, he or she puts coins into slots at one of two stations and types in the number of washing machines he/she will need. The stations are connected to a central computer that automatically assigns available machines and outputs tokens that identify the machines to be used. The customer puts laundry into the machines and inserts each token into the machine indicated on the token. When a machine finishes its cycle, it informs the computer that it is available again. The computer maintains an array *available[NMACHINES]* whose elements are non-zero if the corresponding machines are available (NMACHINES is a constant indicating how many machines there are in the laundromat), and a semaphore *nfree* that indicates how many machines are available.

The code to allocate and release machines is as follows:

```
int allocate() /* Returns index of available machine.*/
{
    int i;
    wait(mutex)
    wait(nfree); /* Wait until a machine is available */
    for (i=0; i < NMACHINES; i++)
        if (available[i] != 0) {
            available[i] = 0;
            signal(mutex);
            return i;
        }
}

release(int machine) /* Releases machine */
{
    wait(mutex);
    available[machine] = 1;
    signal(mutex);
    signal(nfree);
}
```

The *available* array is initialized to all ones, and *nfree* is initialized to NMACHINES. Here *mutex* is a semaphore that is initialized to 1.

I contend that there is a logical flaw in the code above. Describe the problem that can occur in this situation, and describe any modifications you would make to the code above in order to eliminate the problem. (5)

A1.

Q2 {30 minutes} Solaris implements multiple-reader, single-writer locks that allow many threads simultaneous read-only access to a protected object. These locks also allow a single thread write access to the object while excluding any readers. The following functions are provided to users.

- `rwlock_init(rw_lock_type *rwlp)`
This function initializes the read/write lock and sets its state to unlocked.
- `rw_rdlock(rw_lock_type *rwlp)`
This function acquires a read lock. It blocks if a writer holds the lock.
- `rw_wrlock(rw_lock_type *rwlp)`
This function acquires a write lock. It blocks if a writer or any readers hold the lock.
- `rw_unlock(rw_lock_type *rwlp)`
This function unlocks a read/write lock if the caller holds either the read or write lock. Note that if multiple readers hold a read/write lock, only the *last reader actually unlocks* the read/write lock.

Implement read-write locks using semaphores. Assume the existence of a semaphore data type. The operations available for using semaphores are the classical `wait()` and `signal()` operations. Assume that there is also a function

`semaphore_init(semaphore_type *sem, int initial_value)`

which initializes the integer component of the semaphore `sem` to `initial_value`.

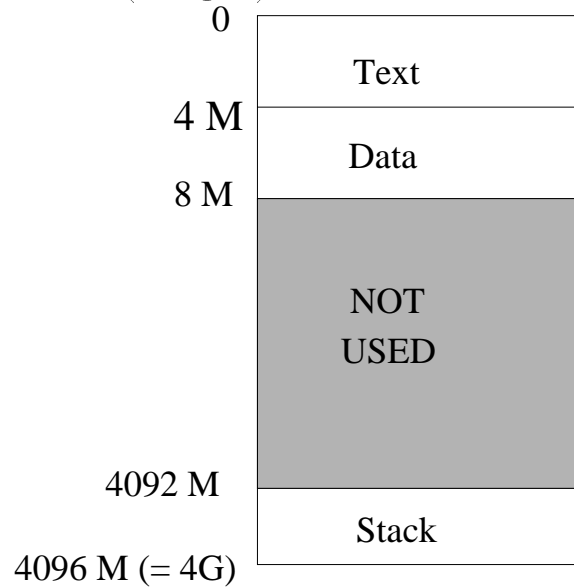
You have to describe the state that needs to be maintained for each read/write lock (e.g., variables associated with each read/write lock, etc.) and write pseudo-code for the functions above. **(20)**

A2.

Q3 { 5 minutes } Five jobs are waiting to be run with expected run-times of 9, 6, 3, 5, and X . Assume that the jobs do not do any I/O. In which order should they be run in order to minimize average turn-around time? In which order should they be run in order to maximize CPU utilization? (Your answer to these questions may depend on the value of X .) **(5)**

A3

Q4 20 minutes Consider a process P executing on a computer with a 32-bit address. Suppose that this process uses only 12 MB of its virtual address space (which has 2^{32} bytes = 4 Gigabytes). The top 4 MB of its address space is used for program text, the next 4 MB for data, and the bottom 4 MB of its address space for the stack (see figure). The addresses in between are not used.



Suppose that the memory is divided into pages of size 4 KB. If a single-level page table is used for this computer, how many entries will the page table have? Design a two-level paging scheme for this architecture, and show how it will reduce the amount of memory required for the page table for process P . [HINT: You have to partition the 32 bit address into three fields – one used as a page offset, one as an index to the outer page table, and one used as an index to an inner page table] (20)

A4.

Q5 {15 minutes} Consider the SITE Lab, where the server computer called **reactor** has a local filesystem (i.e., a local disk) mounted on the directory */usr*. Now assume that the machine **zinc** in the same Lab has this filesystem (i.e., */usr* on **reactor**) NFS mounted on the directory */faculty*. Describe the actions taken by the VFS layer of the UNIX kernel and the NFS client on **zinc** when a process on **zinc** issues a request to open the file */faculty/setia/exam.tex*. **(10)**

A5.

Q6 {15 minutes} Expand on the following statements:

1. Because of its stateless nature, NFS cannot guarantee UNIX semantics.
2. The stateless nature of NFS has a negative impact on performance.

(15)

A6.

Q7 {15 minutes} What “state” is maintained by the AFS file server? What are the advantages and disadvantages of maintaining this state? **(10)**

A7.

Q8 {15 minutes} What is meant by *at most once* and *at least once* RPC semantics? How can the underlying RPC protocol provide *at least once* semantics? How can it provide *at most once* semantics?
(15)

A8

