

CS 385—Test 2—Solution

Instruction: Jon A. Solworth

26 Nov. 2003

1	2	3	4	5	6	7	8	9	10

Name/SSN: _____

Short questions: answer on exam sheet.

1. (10pts.) Using semaphore **unix** system calls write two program fragments. The program fragments should alternate forever executing `func0()` followed by `func1()` (followed by `func0()` followed by `func1()` ...). The code for `func0` is in `program0` and the code for `func1()` from `program1`.

```
sem1 = sem_open(“/sem1”, ORDWR|O_CREAT, 1);
sem2 = sem_open(“/sem2”, ORDWR|O_CREAT, 0);
```

`program0`:

```
while(1) {
    sem_wait(sem1);
    func0();
    sem_post(sem2);
}
```

`program1`:

```
while(1) {
    sem_wait(sem2);
    func1();
    sem_post(sem1);
}
```

2. (15pts.) Using only **unix** system calls create some shared memory to hold i (a 32 bit integer) and create a semaphore for mutual exclusion. Then write a routine which performs $i++$ atomically.

```

shmId = shm_open(“/shm”, ORDWR|O_CREAT, S_IRWXU);
ftruncate(shmId, sizeof(int));
i = (int *) mmap(NULL, sizeof(int), PROT_READ|PROT_WRITE,
                 MAP_SHARED, shmId, 0);

mutex = sem_open(“/mutex”, ORDWR|O_CREAT, 1);

increment()
{
    sem_wait(mutex);
    (*i)++;
    sem_post(mutex);
}

```

3. (12 pts.) Bankers Algorithm. Consider the following state of the Banker’s Algorithm. Is it safe to allocate 1 unit of R_1 to P_1 ? Show all work.

	R_1	R_2	R_3	R_4	finished
P_1	0/2	1/2	0/1	0/0	false
P_2	0/0	0/1	0/0	1/1	false
P_3	1/1	1/1	1/1	1/2	false
total resources	2	3	2	3	
available resources	1	1	1	1	

ans. The configuration before the allocation is safe, as shown by finishing in order P_2, P_3, P_1 . After the allocation we have:

	R_1	R_2	R_3	R_4	finished
P_1	1/1	1/2	0/1	0/0	false
P_2	0/0	0/1	0/0	1/1	false
P_3	1/1	1/1	1/1	1/2	false
total resources	2	3	2	3	
available resources	0	1	1	1	

Now in the safety simulation P_2 can finish resulting in:

	R_1	R_2	R_3	R_4	finished
P_1	1/1	1/2	0/1	0/0	false
P_2	0/0	0/0	0/0	0/0	true
P_3	1/1	1/1	1/1	1/2	false
total resources	2	3	2	3	
available resources	0	1	1	2	

Unfortunately, neither P_1 nor P_3 can finish and hence the state is *unsafe*.

4. (8pts.) Consider a system with n resource types R_0, \dots, R_{n-1} with n units of each resource.

(a) How would you do resource allocation and prevent deadlock by avoiding circular wait?

ans. Allocate the resources in increasing order of subscript.

(b) How would you do resource allocation and prevent deadlock by avoiding hold and wait?

ans. Several possibilities exist, for example: 1. Request all resources at once -or- 2. Release all resources before requesting any more -or- 3. If you cannot acquire a resource release all your resources.

5. (8pts.) Deadlock detection.

(a) **2pts.** Under what conditions can you use the Wait-for graph to detect deadlock?

ans. When each resource has a single instance.

(b) **2pts.** Under what conditions can you use the Banker's Algorithm to detect deadlock?

ans. When resources have arbitrary number of instances.

(c) **4pts.** How would you use the Banker's Algorithm to detect deadlock?

ans. The available resources are those that have not been allocated. The allocated resources are those that have been allocated to the process. The need array contains the resources requested which have not been granted. If the resulting configuration is unsafe then the system is in deadlock.

6. (11pts.) List **all** the steps that happen on a trap instruction when there are base and limits registers?

ans.

- *save the program counter plus other state*
- *set the privilege bit*
- *set base to 0 and limit to ∞*
- *jump to the address specified by the trap number entry in the trap vector.*

7. (12pts.) Two-phase locking.

(a) List the steps (including conditions) necessary to obtain a read lock.

ans. if not write lock then increment the lockCount;

(b) List the steps (including conditions) necessary to obtain a write lock on a location which is not currently locked.

ans. if unlocked then set lockCount to WriteLock;

(c) List the steps (including conditions) necessary to upgrade a lock on a location for which the upgrade can be performed.

ans. if lockCount=1 and if process has a read lock than set lockCount to WriteLock.

8. **(6 pts.)** Little's law. If a bank wants to ensure that customers do not wait on average more than five minutes, when should a new teller be added?

ans. Little's law: $n = \lambda w$ and $w = 5 \text{ minutes}$. Solving for w we get $w = \lambda/n$. We should add resources whenever w gets larger than 5 minutes, ie $5 \text{ minutes} \leq \lambda/n$

9. **(8pts.)** Scheduling.

(a) When does a process get moved to the wait state?

ans. When it initiates an operation with latency in the kernel.

(b) How does a process get removed from the wait state, and what state does it move to?

ans. When the event it was waiting on occurs, at which point it is moved to the ready state.

10. **(10pts.)** Write monitor code to implement **insert** and **delete** on a linked list.

```
class List {
    int value;
    List *next;
};

monitor {
    private:
        List *headPtr=NULL;

    public:
        void
        insert(int v) {
            List *e = new List;
            e->value = v;
            e->next = headPtr;
            headPtr = e;
        }

        void
```

```

delete(int v) {
    List *p=headPtr->next;
    List *l=headPtr;

    if (headPtr->value == v) {
        headPtr=headPtr->next;
        return;
    }

    while (p->value != v) {
        if (p->next==NULL) return;
        l=p;
        p=p->next;
    }
    l->next = l->next->next;
    delete p;
}
}

```