

CS 385—Test 1

1 Apr 2004

Instructor: Jon A. Solworth

Name/SSN: _____

Short questions: answer on exam sheet.

1. **Unix Programming (12pts.)** Using only system calls such as read, write, fork, exec, dup, pipe, write a program fragment which will create a pipe and a child process. The child should write to the pipe “hi there” and the parent will read the string. The child process should terminate and the parent should check its status.

```
int fd;
pid_t child;
char buf[1024];

if (pipe(fd) < 0) {
    printf("Error_creating_pipe");
    exit(0);
}

if ((child = fork()) < 0) {
    printf("Error_creating_child");
    exit(0);
}

if (child == 0) {
    char *str = "Hi_There!!";
    close(fd[0]);
    write(fd[1], str, strlen(str) + 1);
    exit(0);
} else {
    close(fd[1]);
    read(fd[0], buf, 1024);
    wait(NULL);
}
```

```
}
```

2. **Unix Programming (12pts.)** Using only system calls such as read, write, fork, exec, dup, pipe, write a program fragment which creates a new process which will execute the program “ls >out”. (The “>out” means that the file “out” replaces standard output before the “ls” is executed.). Note that the executable (“ls”) and the output file (“out”) are *not* parameters, they are fixed for the program fragment.

```
char *prog = "ls";
char *filename = "out";
pid_t child;

if ((child = fork()) < 0) {
    printf("Error_creating_child");
    exit(0);
}

if (child == 0) {
    int fd = open(filename, ORDWR|O_CREAT, S_IRWXU);
    if (fd < 0)
        exit(0);
    close(1);
    dup(fd);
    if (execlp(prog, NULL) < 0)
        exit(0);
}

wait(NULL);
```

3. **Unix Programming (12pts.)** Using only system calls write a program fragment that maps an existing shared memory object `shmList`, and initializes the linked list in shared memory. Each node in the list contains two fields – value (int) and next. The constant `listCount` contains the number of nodes. Assume that this is the only process updating the shared memory object.

```
struct node {
    int value;
    int next;
} *nodePtr;

int shmId = shm_open("/shmList", ORDWR);
nodePtr = (struct node *)
    mmap(NULL, PROT_READ|PROT_WRITE, MAP_SHARED, shmId,
```

```

        listCount*sizeof(struct node), 0);

    for (int i=0; i < listCount; i++) { /* initialize */
        nodePtr[i].next = i+1;
        nodePtr[listCount-1].next = -1; /* -1 for NULL */
    }

```

4. **Semaphores (12 pts.)** Consider a game in which each player p can either

- increase the p 's position by 1 (`increment(int player)`) or
- decrease the p 's position by 1 (`decrement(int player)`), to a minimum of 0.

(Presumably the player either increases her position or decreases another players position.) The first player to reach 10 wins and only that player should set the variable `winner` to the player number. Implement `increment(int player)` and `decrement(int player)` using semaphores, so that each player's position can concurrently be updated and exactly one process writes `winner`.

```

// shared variables
int winner = -1;           // no winner yet
int position[PlayerCount]; // all zeros
semaphore mutex[PlayerCount] = 1; // general purpose
semaphore winnerMutex = 1;

```

```

void
increment(int player)
{
    wait(mutex[player]);
    position[player]++;
    if (10==position[player]) {
        wait(winnerMutex);
        if (-1==winner)
            winner=player;
        signal(winnerMutex);
    }
    signal(mutex[player]);
}

```

```

void
decrement(int player)
{
    wait(mutex[player]);
    if (position[player]>0)

```

```

        position [player]--;
        signal(mutex[player]);
    }

```

5. **Semaphores (10pts.)** Consider a concurrent system with two queues, q1 and q2. Concurrent inserts should be allowed on q1 and q2, but dequeue should remove an element from q1 (if q1 is not empty) otherwise from q2 (if q2 is not empty). Implement a semaphore based program the routines `insert` and `remove`.

```

insert(int queue, Elmt *elmt)
{
    if (1==queue) {
        wait(sem1);
        q1.append(elmt);
        signal(sem1);
    } else {
        wait(sem2);
        q2.append(elmt);
        signal(sem2);
    }
}

```

```

Elmt*
remove()
{
    Elmt *elmt;
    wait(sem1);
    if (!q1.empty()) {
        elmt = q1.dequeue();
        signal(sem2);
        return elmt;
    }
    signal(sem1);

    // q1 empty, try q2
    wait(sem2);
    if (!q2.empty()) {
        elmt = q2.dequeue();
        signal(sem2);
        return elmt;
    }
    signal(sem2);
}

```

```
    return NULL;
}
```

6. **Deadlock (12pts.)** Describe whether deadlock is avoided or not. If avoided, describe what condition of Dijkstra's is violated to prevent deadlock. If not avoided, give an example of deadlock.
- (a) Resources are A, B, C. Request any subset of the resources in the order A, C, B.
ans. Avoids deadlock by preventing circular wait.
 - (b) Resources are numbered 1,2,3,... Request odd resources before even resources.
ans. Can deadlock if one process requests 1,3 and a second process requests 3,1.
 - (c) If a process requests a resource, and the resource is unavailable, one of the resources the process currently has is returned to the system, and the process must repeat the request again.
ans. Avoids deadlock since it (eventually) denies hold and wait.
 - (d) Every 10 minutes all the processes in the system are killed.
ans. Avoids deadlock through preemption. Note that deadlock is an indefinite wait.
7. **Architectural Support for OS (12pts.)** Each of the following are special purpose processors registers. For each, describe its purpose and when it changes.
- (a) Privilege Bit
ans. Ensures that processes cannot violate the process model by requiring those instructions necessary to manipulate processes to be executed only in privilege mode. (Privilege mode is off while executing processes and on when executing in the kernel.) It is set on processor initialization, on traps, and on interrupts. It is cleared by rtt and rti.
 - (b) TrapVectorPtr
ans. Used by trap instructions to ensure that the kernel is only entered at well defined entry points. Set at kernel load time.
 - (c) base
ans. ensures process does not access memory before the process is located. Changed when entering the kernel and when returning to the process.
 - (d) limits
ans. ensures process does not access memory after the process is located. Changed when entering the kernel and when returning to the process.
 - (e) IntrVectorPtr
ans. points to a vector which specifies entry points (addresses) in the kernel to begin execution for each interrupt type. Changes only on system initialization

(f) InterruptMask

ans. allows interrupts to be turned off when updating a data structure that an interrupt might also update. Interrupt masking is set by privilege instructions.

8. **Interference (8pts.)** What are the possible outcomes of interleavings, assuming all variables are shared and initially 0 in the below questions. Show the value of the variables at completion and an instruction interleaving which achieves each set of values.

- Each of two processes execute the sequence

```
i +=1;
j=i ;
```

Which is compiled into 5 machine language instruction (load-increment-store,load-store)?

ans. process a:(a1,a2,a3,a4,a5) and process b:(b1,b2,b3,b4,b5)

i = j = 2 a1,a2,a3,a4,a5,b1,b2,b3,b4,b5

i = j = 1 a1,b1,b2,b3,b4,b5,a2,a3,a4,a5

i = 2, j = 1 a1,a2,a3,a4,b1,b2,b3,b4,b5,a5

- One process executes $x = 1; y = x$ and the other process $z = y; x = y$?

ans. process a:(a1,a2,a3) store, load, store and process b:(b1,b2,b3,b4) load, store, load, store

x = 0, y = 0, z = 0 b1,b2,a1,b3,b4,a2,a3

x = 1, y = 1, z = 1 a1,a2,a3,b1,b2,b3,b4

x = 1, y = 1, z = 0 b1,b2,a1,a2,a3,b3,b4

x = 0, y = 1, z = 0 a1,b1,b2,b3,a2,a3,b4

9. **Implementing Processes (10pts.)** In what ways might a process try to interfere with another process and what is the technique by which it is prevented?

reading or writing another process's or the kernel's memory: Base and limits registers.

infinite loop timer interrupt

take control over some resource resource allocation is controlled by the OS, and those resources that do not have to be allocated exclusively to a process are managed by the OS.