Q1	Q2	Q3	Q4	Tot

CEng 536 - Advanced Unix

Fall 2016-2017

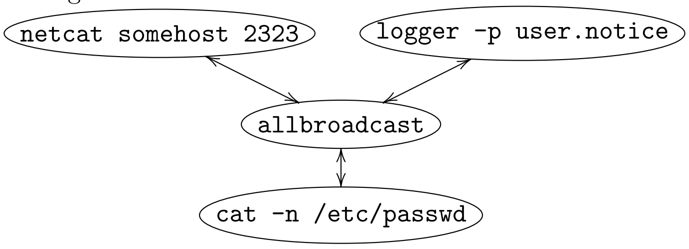
Take home midterm (8 pages, 4 questions, 115 points)

Name: ____________________ **No:** ________ **Signature:** ______________

Note: You are not expected to give complete codes in any of the questions. Just provide significant and critical parts of the code. Your code is not expected to work, just describe primary parts of such a program.

QUESTION 1.(25 points)

Write core part of a function allbroadcast that uses Unix pipes to build a pipe based all to all communication among user commands.



allbroadcast will fork-exec all of its arguments and establish a one-to-one bidirectional pipe with each. Standard input and output of commands will be directed to this pipe. (Assume pipe() creates bidirectional pipes). Then it will read data comming from any pipe and write to all other (not the sender) processes. In the example cat output is sent to logger and netcat, netcat output is sent to cat and logger, etc. (what programs do with this input is not important). Use select () or poll () for blocking on all descriptors. Make reads of size 100 at a time.

Only provide the crucial parts, creating/executing processes, input/output direction, read and write. Assume exec() family functions gets the command string with whitespaces as arguments.

```
const char *comm[]{"/usr/bin/netcat somehost 2323", "/bin/logger -p user.notice",
        "/bin/cat -n /etc/passwd", NULL};
int allbroadcast(const char *commands[]) {  /* Assume commands terminated by NULL */













}
```

QUESTION 2.(30 points)

Condition variables are useful however pthread library can block only on one condition variable at a time. Assume you need sets of condition variables and a program can wait on an arbitary subset of conditions. A sample code is given:

```
pthread_mutex_t mut;
...
class CVset myset(4, &mut);    /* initializes a set of variables from 0 to 3 */


/* thread 1 */
short set[4] = {1,0,1,0};

myset.lock();                        /* lock the mutex given in constructor */
while (!ready[0] || !ready[2]) {    /* ready can be anything */
        myset.wait(set);             /* wait until any of 0 or 2 are notified */
}
...
myset.unlock()
...

/* thread 2 */
short set2[4] = {0,1,1,1};

myset.lock();
while (!complete) {
        myset.wait(set2);     /* wait until any of 1, 2, or 3 are notified */
}
...
myset.unlock();


/* thread 3 */


myset.signal(2);          /* this signal will unblock either of thread1 and thread 2 *
```

Assuming you have all data structures you need, provide an implementation of CVset class. Use pseudo-code whenever appropriate. Provide class definition, wait() and signal () methods. Hint: create a new condition variable per wait() request (to be more efficient you can create per distinct request instead) and block on that. Signaller choose whom to wake up and signal that.

```
class CVset {
```

3

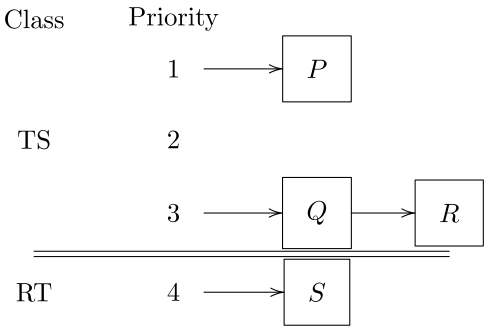
```
}
```

QUESTION 3.(30 points)

a) Assume you have a Unix version implementing real-time and time-sharing scheduling classes. Real-time class uses a single priority level which is 4, and time-sharing class uses three priority levels, 1-3. 4 is the highest priority and 1 is the lowest one. Real-time class always gives 50 ticks as quantum of the process. The priority dispatch table for time-sharing class is given as (all times are in units of ticks in milliseconds, 1/1000 of a second):

Priority	Quantum	Expired pri.	After sleep pri.	Max wait (ticks)	Pri. after Max wait
1	90	1	2	200	2
2	60	1	3	200	3
3	30	2	3	200	3

System has four processes: P, Q, R and S. Current state of the priority dispatch queue is as follows:



Assume instructions in the processes are simplified as follows:

P	*Q*	*R*	*S*
10 ms CPU	60 ms CPU	60 ms CPU	10 ms CPU
50 ms sleep	20 ms sleep	10 ms sleep	20 ms sleep
70 ms CPU	20 ms CPU	50 ms CPU	10 ms CPU
terminate	terminate	terminate	20 ms sleep
			10 ms CPU
			terminate

Give states of processes in this system until all processes terminate in the following table:

t **(10 ticks)**	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
P																														
Q																														
R																														
S																														

Leave cell empty if process terminated. Otherwise fill cell with:

`R`	Running on CPU
`W`	Ready, waiting for CPU
`S`	Sleeping

Mark all context switches on a boundary and mark the reason as one of:

- Expired
- Slept
- Preempted

Please make the following assumptions:

- Processes with expired quantum are inserted at the end of their new priority queue.
- When processes wake up from sleep they are inserted at the end of their new priority level queue and preempt the process in the lower priority if CPU is occupied by such a process.
- The preempted processes are inserted at the beginning of their current priority queue.
- When a preempted process takes CPU again its quantum starts from where it is left, not reset.

b) Assume only P, Q, R exists in previous section in a Linux system. Give states of processes in this system until all processes terminate in the following table:

t **(10 ticks)**	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
P																														
Q																														
R																														
S																														

Please make the following assumptions:

- A fixed quantum, 40 ticks is given to the running task. No runnable task preempts the running task during that period.
- A waking up task may preempt the running task. Preempted tasks quantum starts from where it is left, not reset.
- Running time of the tasks are not weighted, their execution time is directly added to their run time.
- Assume tasks are started in order P, Q, R and inserted in the data structure with this order in case they have equal running time values.

QUESTION 4.(30 points)

Provide simple kernel codes (just data structure detail, no locking, syncronization, error checking etc.) that will do the following tasks:

a) **Linux** code that will traverse system process table and set pid to process id and ppid to parent proces id in the loop body. As:

```
for (           ;               ;            ) {
                  int pid =                                 ;
                  int ppid =                                ;
                  ....
}
```

Hint: init_task

b) **OpenSolaris** code that will traverse system process table and set pid to process id and ppid to parent proces id in the loop body.
Hint: practive

c) Write a **Linux** function ptree(p)that recursively traverse all descendants (children of all depths) of a process task_struct *p and calls do_something(task_struct *c) for the child task. The traversal order is not important.

d) Write a **OpenSolaris** function ptree(p) that recursively traverse all descendants (children of all depths) of a process proc_t *p and calls do_something(proc_t *c) for the child task. The traversal order is not important.
e) What is the entry of fork system call in **Linux** kernel. Just provide the filename and function name.
f) What is the entry of fork system call in **Solaris** kernel. Just provide the filename and function name.
g) A simplified **Linux** code that will make fork(p) for a struct task_struct *p process and return the child process. Only provide fields related to

- Credentials
- pid/parent/child/sibling relations
- file descriptor table and file structures

Use kmalloc(sizeof(...)) to allocate a new structure. Only provide relevant fields. Use pseudo code when you need to insert/delete a value in a data structure.
h) A simplified **Solaris** code that will make fork(p) for a proc_t *p process and return the child process. Only provide fields related to

- Credentials
- pid/parent/child/sibling relations
- file descriptor table and file structures

Use kmalloc(sizeof(...)) to allocate a new structure. Only provide relevant fields. Use pseudo code when you need to insert/delete a value in a data structure.