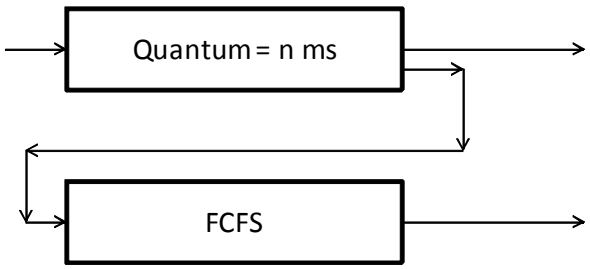
NIA: STUDENT NAME:

1. **(3pt)** A system with one CPU employs a two level feed-back queue scheduling algorithm. First, a process is scheduled for a quantum of *n* milliseconds and, if not finished, by using the FCFS policy, as shown in the image:



Given the following set of processes:

Process	Burst Time (ms)	Arrival time
*P*1	9	0
*P*2	3	1
*P*3	1	3
*P*4	2	2

a. In the following Gantt charts illustrate the execution of the four processes using the two-level scheduling policy for four cache cases: ***n=1 ms***, ***n=3ms, non-premptive SJF and FCFS***.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
n=1	1	2	4	3	1	1	1	1	1	1	1	1	2	2	4
n=3	1	1	1	2	2	2	4	4	3	1	1	1	1	1	1
SJF	1	1	1	1	1	1	1	1	1	3	4	4	2	2	2
FCFS	1	1	1	1	1	1	1	1	1	2	2	2	4	4	3

b. What is the turnaround and waiting time of each process for each of the scheduling algorithms in part a?

Turnaround time	P1	P2	P3	P4
n=1	12	13	1	13
n=3	15	5	6	6
SJF	9	14	7	10
FCFS	9	11	12	12

Waiting time	P1	P2	P3	P4
n=1	3	10	0	11
n=3	6	2	5	4
SJF	0	11	6	8
FCFS	0	8	11	10

c. How many context switches are performed for each of the scheduling algorithms in part a?

n=1: 6
n=3: 4
SJF:3
FCFS:3

d. Assuming that a context switch takes 0.1 ms, what is the CPU utilization for the scheduling algorithms in part a?

n=1: 14/14.6
n=3: 14/14.4
SJF: 14/14.3
FCFS: 14/14.3

e. Compare the four scheduling algorithms. Which are the advantages and disadvantages of each one?

See Silberschatz Chapter 5.

2. (1pt) Which are the advantages and disadvantages of the microkernel approach to system design?

Answer: Benefits typically include the following: (a) adding a new service does not require modifying the kernel, (b) it is more secure as more operations are done in user mode than in kernel mode, and (c) a simpler kernel design and functionality typically results in a more reliable operating system. User programs and system services interact in a microkernel architecture by using interprocess communication mechanisms such asmessaging. Thesemessages are conveyed by the operating system. The primary disadvantages of the microkernel architecture are the overheads associated with interprocess communication and the frequent use of the operating system's messaging functions in order to enable the user process and the system service to interact with each other.

3. (1pt) Describe the differences among short-term, medium-term, and long-term scheduling.

Answer:
a. Short-term (CPU scheduler)—selects from jobs in memory those jobs that are ready to execute and allocates the CPU to them.
b. Medium-term—used especially with time-sharing systems as an intermediate scheduling level.Aswapping scheme is implemented to remove partially run programs frommemory and reinstate them later to continue where they left off.
c. Long-term (job scheduler)—determines which jobs are brought into memory for processing.
The primary difference is in the frequency of their execution. The short term must
select a new process quite often. Long-term is usedmuch less often since it handles placing jobs in the system and may wait a while for a job to finish before it admits another one.

4. (1pt) Describe the actions taken by a kernel to context-switch between processes.

Answer: In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

5. (1pt) Define a system call and explain how system calls are typically implemented in operating systems. Which are the existing methods for passing parameters to system calls?

Answer: System calls represent the interface of operating systems. The system calls intercept function calls to be served by the operating systems and invoke the necessary calls inside the kernel. For this typically the executing mode is changed from user mode to kernel mode. Each system call is associated a number, which may be used as an index into a table containing the addressed of the implemented functions. Typical methods of parring parameters to system calls include: a. Pass parameters in registers b. Registers pass starting addresses of blocks of parameters c. Parameters can be placed, or pushed, onto the stack by the program, and popped off the stack by the operating system

6. (3pt) Develop a small program in C using system calls that does the following. A father P1 creates two children P2 and P3. Each child creates two grandchildren: P2 creates P4 and P5 and P3 creates P6 and P7. Each grandchild sends to its parent its pid ***through a common file***. Subsequently P2 and P3 sum the pids of their corresponding grandchildren and return the sum

to the process P1. P1sums the value returned by P2 and P3 and, finally, prints the result. Your program must avoid the creation of zombies.

Useful functions:

I/O

```
int close(int fd)
int open(const char *path, int oflag)
int write(int fd, void *buf, int nbyte)
ssize_t read(int fd, void *buf, size_t nbyte)
int lseek(int fd, int offset, int whence)
```

Process Management

```
int wait(int *status)
pid_t fork(void)
void exit(int status);
```

Answer:

```
int main()
{
  char fname[20];
  pid_t pid;
  int status,i,j,fd[3], sum;

  for (i=0;i<3;i++) {
    sprintf(fname, "f%d",i);
    fd[i]=open(fname,O_CREAT|O_RDWR,0666);
  }
  for (i=0;i<2;i++){
    if(fork()==0) {
      sum=0;
      for (j=0;j<2;j++) {
            if(fork()==0){
                  pid = getpid();
                  write(fd[i],&pid,sizeof(int));
                  exit(0);
            }
            else{
                  wait(&status);
                  lseek(fd[i],j*sizeof(int),SEEK_SET);
                  read(fd[i],&pid,sizeof(int));
                  totalpid+=pid;
            }
      }
      write(fd[2],&totalpid,sizeof(int));
      exit(0);
    }
  }
  sum = 0;
  for (i=0;i<2;i++) {
    wait(&status);
    lseek(fd[2],i*sizeof(int),SEEK_SET);
    read(fd[2],&pid,sizeof(int));
    sum+=pid;
  }
  printf("sum = %d\n",sum);
  return 0;}
```