OPERATING SYSTEM PROJECT: SOS

I. Description

1. This project simulates a noninteractive (batch) monolithic operating system. Your program, OS, is a set of functions invoked by SOS (Student Operating System), an existing program. SOS creates a series of simulated user programs and simulated requests, and invokes the OS functions to "handle" them. **NOTE**: in reality, there is no job stream; this is a simulation!

2. The following (simulated) hardware is available in the SOS system:
   a) A CPU.
   b) A 100K word main memory, allocated in units of 1K words.
   c) A disk which holds simulated I/O files.
   d) A drum which is used to swap files in and out of memory.
   e) An IntervalTimer: a register with load and decrement operations, and which can generate an interrupt. The time slice (the maximum time a job can run in one CPU burst) can be loaded in OS by setting variable p[4], representing time in ms (milliseconds). In SOS, the IntervalTimer decrements once for each ms of processing time. When it reaches 0, the SOS calls Tro (Time Run-out), a function in OS.
   f) A real-time clock: a many bit register which increments for each millisecond that a user job is running or the CPU is idle. Accessed in OS using variable p[5], the real-time is the current time.
   g) An job spooler which feeds the (simulated) system drum with new (simulated) jobs.

** The SOS program was originally developed by E. Akkoyunlu in PL/1 **

II. Structure of OS

1. SOS Calls to OS

a) SOS starts by calling the OS function startup. This gives you a way to initialize variables. You must decide what to put into startup.

header: void startup(void);

b) The following five functions are interrupt handlers, defined in OS. Each has two parameters: int *a and int p[] (6-element integer array). **For some older compilers, all variables described here as intare to be made long**. When SOS invokes an OS interrupt handler, or when the OS function finishes and returns to SOS, information is passed through *a and p.

● Crint: invoked when the job spooler sends a new job to the system (a new job has entered the system and is now on the simulated drum).

header: void Crint (int *a, int p[]); // p is an array of 6 elements; alternate: int *p

When Crint is called by SOS, the values of the parameters are as follows (p[0] is unused):

p[1]: job number p[2]: job priority
p[3]: job size (in kb) p[4]: maximum CPU time
p[5]: current time

● Dskint: Invoked when the disk generates an interrupt (an I/O transfer between the disk and memory has been completed).

header: void Dskint (int *a, int p[]);

When Dskint is called by SOS, p[5] is current time; ignore the rest.

● Drmint: invoked when the drum generates an interrupt (a program transfer between the drum and memory has completed a swap).

header: void Drmint (int *a, int p[]);

When Drmint is called by SOS, p[5] is current time; ignore the rest.

● Svc: Supervisor Call Interrupt, invoked when the executing job needs service and has just executed a software interrupt instruction.

header: void Svc (int *a, int p[]);

When Svc is called by SOS, the values of the parameters are as follows:

p[5]: current time.

*a:
If *a=5: the job is requesting termination.
If *a=6: the job is requesting another disk I/O operation.
If *a=7: the job is requesting to be blocked until all pending I/O requests are completed.

● Tro: invoked when the IntervalTimer's register has decremented to 0. Timer interrupt.

header: void Tro (int *a, int p[])

When Tro is called by SOS, p[5] is current time; ignore the rest.

To Run A Program: Before leaving any of the five interrupt handler functions, the arguments (**int** *a and **int** p[]) must be set as follows:

*a: gives the state of the CPU:

*a=1: no jobs to run, CPU will wait until interrupt (extremely rare situation, but it can come up!)
Ignore p values.

*a=2: set CPU to run mode, must set p values as below.

- p[0], p[1], and p[5]: ignored
- p[2]: the base address of job to be run.
- p[3]: the size (in K) of job to be run.
- p[4]: time slice (time quantum)

2. OS Calls to SOS

These functions (representing I/O operations) are defined in SOS.

a) siodisk: invoked by OS to start a disk data transfer.

header (in SOS): void siodisk(int JobNum);

The argument JobNum is a job number.

b) siodrum: invoked by OS to start a drum transfer (swap).

header (in SOS): void siodrum(int JobNum, int JobSize, int StartCoreAddr, int TransferDir);

Parameters used:
JobNum: job number
JobSize: job size.
StartCoreAddr: starting core address.
TransferDir: specifies the transfer direction:
0 means drum to memory
1 means memory to drum

c) ontrace and offtrace: invoked in OS, turn on or off the tracing mechanism. The default value: OFF.

WARNING: ontrace produces a blow-by-blow description of each event and results in an extremely large amount of output. It should be used only as an aid in debugging. Even with the trace off, performance statistics are generated at regular intervals and a diagnostic message appears in case of a crash. In either case, OS needs to print nothing.

headers (in SOS): void ontrace(void); void offtrace(void);

3. More OS Functions (Called by Other OS Functions Only)

a) Memory Manager: Memory is 100 K words, allocated in units of 1 K and numbered 0-99. At any given time, some words may be free, and some allocated. In order to swap a job into memory from the drum (say, job ComingIn), OS must invoke the memory manager to allocate a block of free memory for the job.

If there is a block of memory large enough for ComingIn:

- the OS memory manager allocates memory to the job ComingIn.
- the OS Swapper (not part of the memory manager), swaps job ComingIn from the drum into this newly allocated memory block.

If there is no block of memory large enough for the job ComingIn:

- the OS calls the memory manager to select (using some algorithm) a job in memory for replacement (say, job Bumped).
- the OS Swapper swaps job Bumped from memory to drum.
- Now, OS Swapper swaps ComingIn from the drum into available memory

When a job terminates, OS invokes the memory manager to deallocate the memory block that was allocated to the job.

b) CPU Scheduler: When the current job stops running, there may be 0 or more jobs ready to run (including jobs which had been interrupted).

If there are jobs waiting to run:

- the OS CPU scheduler selects a job to run on the CPU (using some algorithm) that is ready to run.
- the OS time manager sets p[4] to the desired time slice.
- the OS Dispatcher sets *a=2, and the CPU values needed in p[2] and p[3].

NOTE: In this simulation, only the base address register p[2] and size register p[3] need to be set; in a more realistic simulation, the program counter, CPU mode, and other registers would also be set.)

If there are no jobs ready to run, then obviously, the scheduler does not select any job; the OS Dispatcher will then halt the CPU (*a=1).

c) Time Manager: sets the time slice. You do not want a single job to tie up the CPU, but most jobs should hit an interrupt before a Tro.

d) Dispatcher: sets CPU registers before context switches.

e) Swapper: handles requests for swapping, starts a drum swap (using SOS function Siodrum), handles drum interrupts (Drmint), and selects which job currently on the drum should be swapped into memory.

4. Disk I/O Operations

These functions include handling requests for data transfers to and from the disk, starting a disk operation (using the SOS function Siodisk), and handling a disk interrupt (Dskint).

5. Miscellaneous

a) OS will need many data structures (booleans, stacks, queues, lists, arrays, arrays of records, etc.) One important data structure, the Jobtable, contains all the information about each job in the system.

b) OS simulates the software; SOS simulates the hardware.

c) SOS and OS never run at the same time.

d) SOS contains the Main function, so OS must not contain Main.

e) Each time SOS calls one of the interrupt handlers, the real-time is passed to the OS function in p[5]. In this simulation the real-time clock is incremented when a user program is running or when the CPU is idle. When OS is running, the real-time clock is not incremented (this gives you time to handle interrupts etc. without affecting system performance). Therefore, current time does not change from the moment an OS function is called to when OS executes a return.

f) OS is an event-driven operating system with the following pattern:

```
                  sleeping OS
                       |
   interrupt, OS now running with interrupts disabled
                       |
     OS keeps track of the interrupted job (if any)
                       |
 OS handles the event (which was signaled by the interrupt)
            (Crint, Dskint, Drmint, Tro, Svc)
                       |
              OS schedules the next job
                       |
               OS dispatches the job
                       |
       OS goes to sleep with interrupts enabled
```

g) Each time an OS interrupt is invoked by SOS, before you return to SOS, you must invoke other OS functions to check

- if there is room in memory to swap in another job; if there is, swap in a job.

- if there are jobs ready to run: if there are, OS must schedule one of these jobs.

h) If a job has outstanding disk I/O request(s) and no requests are being serviced by the disk, OS can decide to leave the job in memory or swap it onto the drum. But, before OS initiates an Siodisk for any of the job's I/O requests, the job must be in memory.

i) If a job is to be terminated, and it has outstanding disk I/O request(s), it cannot be killed. OS must wait to kill the job until all disk I/O operations are completed.

j) A terminated job does not have to be swapped out, but its memory and its Job Table entry must be freed for use by another job.

k) You do not need to do any printing except for debugging, since SOS automatically prints a running account of what is going on. The SOS offtrace and ontrace functions turn off/on a very detailed trace. After you have debugged your program, you should turn the trace off at the beginning of the program. You can turn it back on for debugging later in the program simply by using an if statement and testing the value of the current time or the job number at the point at which you want to resume tracing.

l) When SOS issues a Crint (meaning a new job is on the drum ready to enter the system), the job comes in with a priority number in p[2] (1 is highest priority, 5 is lowest priority). You may use or change job priorities in your scheduling, or you may ignore the whole issue.

m) Interrupts are disabled while OS is executing.

III. Notes

1. SOS will throw jobs and service requests at OS, and OS must handle them. SOS will terminate when:
   - the jobs become backed up in the job table (there is room only for 50 jobs)
   - an error is made in handling a job (ex: OS invokes siodisk to request I/O, but the job is not in memory; a job is terminated before all outstanding I/O requests are finished; etc.)
   - SOS runs to completion.

Once the program runs, you must constantly try to increase your efficiency, using common sense and ideas from class, to get SOS to run to completion.

2. In writing OS, you will be concerned with CPU scheduling, time management, Main memory management, swapping, file transfers, and system calls. You should write a modular program: each service, no matter how small, should be handled by a separate function. (For example, the Dispatcher and Time Manager will probably be small, simple functions; nevertheless, they should be separate functions.)

3. Do not make any of your C++ routines so large that they cannot fit on a printed page. The code for each routine should be only a few lines int (10 or 20, maybe 30 at the very outside).
   a) Your C++ code must contain **extensive** comments explaining:
      - what the function does.
      - what data structures and fields are used.
      - how the function works (if the code is difficult to follow).

b) Use meaningful identifiers; they help comment the code.

c) Use meaningful indentation; this helps the reader.

d) If the function of a variable is not clear, rename it. You can also use comments at the start of your program to explain what it does.