Assignment 2

Due October 21^{st} 2009

Discrete simulation. Modelling input.

The problem

A simple computer system runs under the control of a simple OS. Two very long processes are currently in execution; they will not terminate in the near future and no other processes are going to appear.

The two processes are **A** and **B**:.

```
B.0     mf = open( ”order” , O_RDONLY , mode ) ;
B.1     inp = open( ”input” , O_RDONLY , mode ) ;
B.2     out = open( ”output” , O_WRONLY , mode ) ;
B.3     for( ;; ) {
B.4         n = read( mf , &offset , sizeof(off_t) ) ;
B.5         if( n != sizeof(off_t) ) break ;
B.6         read( inp , buf , BLOCK ) ;
B.7         lseek( out , offset , 0 ) ;
B.8         write( out , buf , BLOCK ) ;
```

Meaningful parameters

T_i: CPU time needed to handle an interrupt (without a context switch). Equal to 200 *ns*.

Included in T_i is all the processing of the interrupt: page–fault handling includes the time needed to pass a read request to the Disk Controller.

T_c: context switch time. Equal to 300 *ns* for the complete switch (store old context or load new context).

T_s: seek time needed by the disk arm. The cylinder number of the previous disk operation is remembered in p (initialise to 900 at the start of simulation).

The seek time is computed as follows. The request is for cylinder number c. Compute $\mathrm{d} = \left| \mathrm{c} - \mathrm{p} \right|$ followed by $\mathrm{p} = \mathrm{c}$. Then (think of d as the distance in cylinders):

$$\mathrm{T}_s = \begin{cases} 0 & \mathrm{d} = 0 \\ 8 \times 10^5 & 0 < \mathrm{d} \leq 3 \\ 8 \times 10^5 + 2 \times (\mathrm{d} - 3) \times 10^5 & 3 < \mathrm{d} \leq 9 \\ 2 \times 10^6 + (\mathrm{d} - 9) \times 10^5 & 9 < \mathrm{d} \leq 19 \\ 3 \times 10^6 + 2 \times (\mathrm{d} - 19) \times 10^4 & \mathrm{d} > 19 \end{cases}$$

T_t is the total transfer time for a sector of size S (one i/o operation). $\mathrm{T}_t = S/0.15$ where the mysterious 0.15 is the transfer rate of 150 MB/s (SATA).

Disk locations

This disk is made of 1800 cylinders of 512 MB each (there are several tracks per cylinder but this in of no relevance here).

The locations of the various objects stored on disk are known:

B the virtual–memory copy of it occupies cylinders 200 to 215.

order occupies part of cylinder 725. It is 2^{25} bytes long.

inp occupies cylinders 800–803 for a total of 2^{22} sectors.

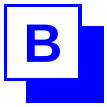
out This dumb system allocated for it space on cylinders 1200–1205 (we know that only 1200–1203 will be used).

Its behaviour is perfectly clear at this point. To simplify matters, from now on the fetch of **A[i][k]** never causes a page fault, so that only **B[k][j]** causes one.

The new aspect is that now we know where the requested pages are . The cylinder numbers referenced form a loop repeated many, many times.

200,200,201,201,202,202, ... , 215 , 215 , 200, ...

etc.



It creates a “shuffled” version of a file using a sequence of keys stored in yet another file. The two input files are read sequentially, one block at a time (block = sector).

The output file is written out in a randomised manner based on the offset read from file **order**. We have a real sample of offset sequences (see www.cis. ... /2460/data).

You will have to imagine a random variable (i.e. a pdf) which you will use when deciding which cylinder will be the destination of the next block of **out** when written out by process B (the values of the variable **offset**).

Interrupt

(not really part of the OS) we have only 2 types of interrupts (Page Fault and Disk). If both occur simultaneously (a rare case), the Page Fault has higher priority and is accepted while the Disk Interrupt is kept spinning.

We will assume that the interrupt itself stores the PC, PSW and the SP in some safe location; furthemore, we will optimistically assume that this operation is instanteneous.

Spinning on an interrupt

If a disk interrupt is not accepted by the **CPU**, the Disk controller keeps repeating it until successful. While it is doing so, it performs no other action.

Scheduler

gives the **CPU** to the most worthy ready process.

The **NULL** process has the lowest priority and is always preempted. If both **A** and **B** are ready, the Scheduler returns the **CPU** to the process that formally has it (i.e. its VM context is stored in the **CPU**). There always is a process with this property.

If the Scheduler gives the CPU to a process that does not have its context in it, it performs a **full context switch**. Otherwise it does not do any switching at all.

If interrupted, the Scheduler quietly disappears without a trace.

Research work

Once you have your basic simulator running, you may consider doing some research work:

Prefetch: consider bringing two adjacent pages of array **B** at the same time (and avoiding a page fault the next time). Note that the second page may still need a seek, but only sometimes and only to the next cylinder, if at all.

Read–ahead: consider always starting reading the next block of **inp** immediately after you accessed the previous one (i.e. before writing out a block of out). Potential savings are similar to the “Prefetch” case.

Cache: consider not writing out to disk immediately when process B says so but copy it to a 16–block cache memory. Write it out only when the cache is full; when you do so, write one–after–another all the blocks going to the same cylinder. The benefits of this scheme are not easily described.

Self evaluation form

Name: _____ Student ID# _____ Total:

Step	Done correctly	Not done	Other (explanation)
1			
2			
3			
4			
5			
6			