Assignment 2

Due October 21st 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**:

```
A.0  double A[N][N] , B[N][N] , C[N][N] ;
A.1  ... // reads A and B
A.2  for( i = 0 ; i ≤ N ; i++ )
A.3     for( j = 0 ; j ≤ N ; j++ )
A.4        C[i][j] = 0 ;
A.5  for( i = 0 ; i ≤ N ; i++ )
A.6     for( j = 0 ; j ≤ N ; j++ )
A.7        for( k = 0 ; k ≤ N ; k++ )
A.8           C[i][j] += A[i][k] * B[k][j] ;
```

```
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 ) ;
     }
```

Note that only parts of the code of the processes are shown. In order to compile and run they will need a few declarations (and **includes**).

Meaningful parameters

The units used are: 1ns as a unit of time and 1B as a unit of memory,

A few parameters will be needed. While a sample value is associated with each of them, your simulator must treat them as parameters.

S: the size of a disk sector. Default value is 512B.

8: size of a double floating point value = 8B.

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 d =| c − p | followed by p = c. Then (think of d as the distance in cylinders):

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

T_l: the rotational latency time is a uniformly distributed value between 0 and $\frac{1}{120}10^9$ (note that 7200 rpm = 120 rps).

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

Z: the Scheduler takes 40 ns to decide what to do. When it decides, it may call in the Dispatcher (Context Switch) or it may not.

N: is a parameter in process **A** and is a value large enough to have $N^2/8 >$ *memory size*.

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).

Processes

A 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.

B 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**).

The OS

Only small parts of the OS are of interest to us:

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.

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.

Interrupt handler is handled in T_i time and does not perform a context switch. The execution of an interrupt handler is not interruptible; if the disk tries to interrupt the **CPU** when an interrupt is handled, it spins. Each interrupt handler ends with the same two instructions: **Enable Interrupts** followed by a **goto** to the Scheduler. Note that enabling interrupts means that one spinning interrupt will get through at this very moment.

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 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.

Handling a page fault: interrupt + Scheduler.

Handling a disk interrupt: interrupt + Scheduler.

Context switching Processes have a **context** which is stored in the **CPU** when they are running (think in terms of Stack Pointer + Program Counter + Page Table Origin, etc.). The **NULL** process does not need any context, so it runs with the context of the previous process left intact.

Disk request as is obvious by now, each disk request comes with the disk address (just the cylinder number). Whenever a disk request is generated (page fault in process A or in process B) a cylinder number must be generated as well. In process A it is given and so it is in the case of reads form files **inp** or **order**. In the case of writing to file it must be modelled based on the supplied random sample.

Hardware

The system's hardware has these properties:

CPU: fetches its arguments from memory (no cache). All instructions have 0 or 1 memory operands, so that an instruction can generate at most one page fault.

Disk Controller: performs one i/o request at a time.

- When a request comes when the disk controller is busy, it is enqueued inside the disk controller. (strictly FCFS).

- All transfers bypass the CPU (DMA is used).
- It interrupts the CPU when the i/o operation is finished. If that interrupt cannot be delivered, it spins for as long as is needed.

To be done

Your task is to write one simulator for this simplistic system: Its output should contain:

CPU utilisation: defined as the fraction of time that process **A** or **B** is using the CPU.

Disk utilisation: defined as the fraction of time that the disk controller is performing data transfer (which does not include the seek and latency delays).

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.

For each of these research projects, the question is the same:

What is the performance now?

You are expected to do at least one of these projects and no more than two. If you would prefer to do another (more ambitious) project, you will have to talk to me. A project is done correctly if performance results are correct and are (approximately) correctly compared with the basic model.

Submission rules

You will deliver your assignment work in person at a time scheduled in advance (a sign–up sheet will be available). You must bring with a filled copy of the self–evaluation form.

Grading

The assignment is worth 10 marks which are distributed as follows:

	action	marks
1	Scheduler works correctly	2
2	Disk requests never lost	2
3	Disk interrupts handled properly	2
4	**out** requests modelled correctly	2
5	Project 1 done correctly	2
6	Project 2 done correctly	2

This assignment must be full of errors. A reward of 1 mark will be given to the first student who points out a relevant error.

Self evaluation form

Name: _____ Student ID# _____ Total: [ ]

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