A Virtual Disk for Pranali

This assignment is the first step in adding a virtual memory (VM) or a file system (FS) to the Pranali OS. In assignment #7, we created a virtual disk named *Sim_disk*, and provided system calls to perform read and write operations on it. We now provide an I/O interrupt processing capability in the virtual machine to make the disk operation more realistic, and provide a feature to block a process that initiates an I/O operation until its operation completes.

The characteristics of the disk device, e.g., block size, number of blocks, track capacity, would be set at booting time by processing the configuration file. You are permitted to change the characteristics to make the disk both realistic and contemporary.

The disk will be represented by a simple *disk model* that calculates the I/O wait time for a read or write operation performed on the disk as a function of the number of tracks by which the disk head has to move. The I/O operation should be deemed to be complete only at the end of that much time, hence we need an arrangement to raise an I/O interrupt at that time. This arrangement is as follows: Each instruction in the hypothetical machine is assumed to consume a certain amount of time, say n nanoseconds. Hence we find how many instructions can be completed in the time between the start of an I/O operation on the disk, and its completion. This number of instructions is used to implement I/O interrupts as explained in the following.

You should provide the following interrupt processing arrangement in the virtual machine:

1. The virtual machine will have a counter indicating the number of instructions executed since start of simulation. It would incremented by 1 after every instruction. Its width would be adequate to give a few month's execution time. We will call this counter "instruction no".

2. The virtual machine will keep a queue of "scheduled interrupts". Each entry in the queue will have a tuple: <instruction no, kind of interrupt, details of interrupt > where <details of interrupt> will be specific to the kind of interrupt. This queue will be maintained in the ascending order by instruction count.

3. The virtual machine will simulate occurrence of an interrupt when the value in the "instruction no" counter matches with the count in the first entry in the queue. It will load values from <details of interrupt> in appropriate CPU registers or memory locations at the time of the interrupt.

Key issues in this assignment:

1. Note that a process under Pranali is actually a C program (see assignment #7). This process would make a system call to initiate a read/write operation on the virtual disk *Sim_disk*.

   - Hence we must insert these system calls in the C program that is to be executed as a process.

2. Pranali should "block" the process that initiates an I/O operation until an I/O interrupt indicates that the I/O operation has completed. Hence we should mark an appropriate process state transformation for the process that initiates an I/O operation. You would have to add this feature to the Pranali OS.

3. The C program that is running as a process under Pranali would perform I/O operations of its own. These operations would be performed on the Linux file system, and they would be implemented by system calls that are handled by the host OS. To make the C program's operation under Pranali more realistic, we would like the I/O interrupt and the blocking of that process to occur when it performs its I/O on the Linux file system. This can be achieved as follows:

   - For each I/O statement in the C program:
     After the statement, insert a system call to perform a read
     or write operation on an appropriate disk block in *Sim_disk*.

4. This way, each original I/O statement would be followed by a *Sim_disk* operation, which would block the Pranali process for a certain period of time. This actually happens as follows:

   - Consider an original I/O statement in the C program. The compiler would compile it as a system call to the host OS (i.e., Linux) for implementing the operation on the Linux file system.
   - Since the C program executes as a process under Pranali, this system call would take place during 'execution' of the program in Pranali. It would transfer control to the host OS, i.e., to Linux. Linux will initiate the I/O operation and block the process of the host OS that made the call, which happens to be (the Pranali OS+Pranali processes). Hence the Pranali OS will freeze. After the I/O is done, the Pranali OS will be activated and the system call would return to the Pranali process that issued the system call, i.e., it would return to resume execution of the C program.
   - The C program would now make the system call that was inserted in Step 3. It will transfer control to the syscall handler of Pranali, which will now schedule an I/O interrupt in the Pranali OS, and block the Pranali process. Pranali would now schedule a process for execution. If none exists, the "instruction no" counter would be advanced to the instruction count in the first entry in the

scheduled interrupts list, and the I/O interrupt will be made to occur.

Work to be done in this assignment

Prepare your own version of the Pranali OS, which you will carry through this entire design project. Add the following features to Pranali:

1. Add the I/O interrupt capability in the simulated machine of Pranali.
2. In Pranali, add the feature to block a process that initiates an I/O operation until its I/O operation completes.
3. In assignment #7, you had added the "instruction slice" feature to Pranali. It should be carried over.

It would be useful for each group to split into two halves to perform the first two functions.

Disclaimer:
Right to make additions/modifications to this lab specification is reserved.