2. PROCESS

2.1 Introduction to processes

2.1.1 The Process model

A process is just an executing program, including current values of the program counter, registers, and variables.

Nowadays, multiprogramming (parallel programming) is common process model, which is the rapid switching back and forth among processes.

Process Hierarchies

In MINIX, processes are created by FORK system call, which creates an identical copy of the calling (parent) process. Note that each process has one parent but zero, one, two or more children.

In MINIX, a special process, called *init*, is present on the boot diskette. When it starts running, it reads a file telling how many terminals there are. Then it forks off on new process per terminal. These processes wait for someone to log in. If a login is successful, the login process executes a shell to accept commands. These commands may start up more processes. So, init is the root process through the whole system.

Process States

1. Running (actually using the CPU)
2. Blocked (unable to run until some external event happens, e.g. input waiting)  
   cat file1 | grep tree: in this case, grep process is blocked untile cat process finishes
3. Ready (runnable; temporarily stopped to let another process run)

\*Refer to Fig. 2-2 on p.48

#running -> blocked

In some systems the process must execute a system call, BLOCK, to get into blocked state. In other systems, including MINIX, when a process reads from a pipe or special file (e.g., a terminal) and there is no input available, the process is automatically blocked.

#running -> ready or ready -> running

These transitions are caused by the process scheduler.

First transition occurs when the scheduler decides that the running process has run long enough and it is time for another process to run. You can guess when the latter occurs

#blocked -> ready

It occurs when the external events finishes (e.g., arrival of input)

When a disk interrupt occurs, the system makes a decision to stop running the current process and run the disk process, which was blocked waiting for the interrupt. When the disk block has been read or the character typed, the process waiting for it is unblocked and is eligible to run again.

“scheduler” really means not just processing scheduling, but also interrupt handling and all the interprocess communication as well.

2.1.2. Implementation of Processes

To implement the process model, the operating system maintains a table, called the process table, with one entry per process.

Refer to Fig. 2-4 on p.50

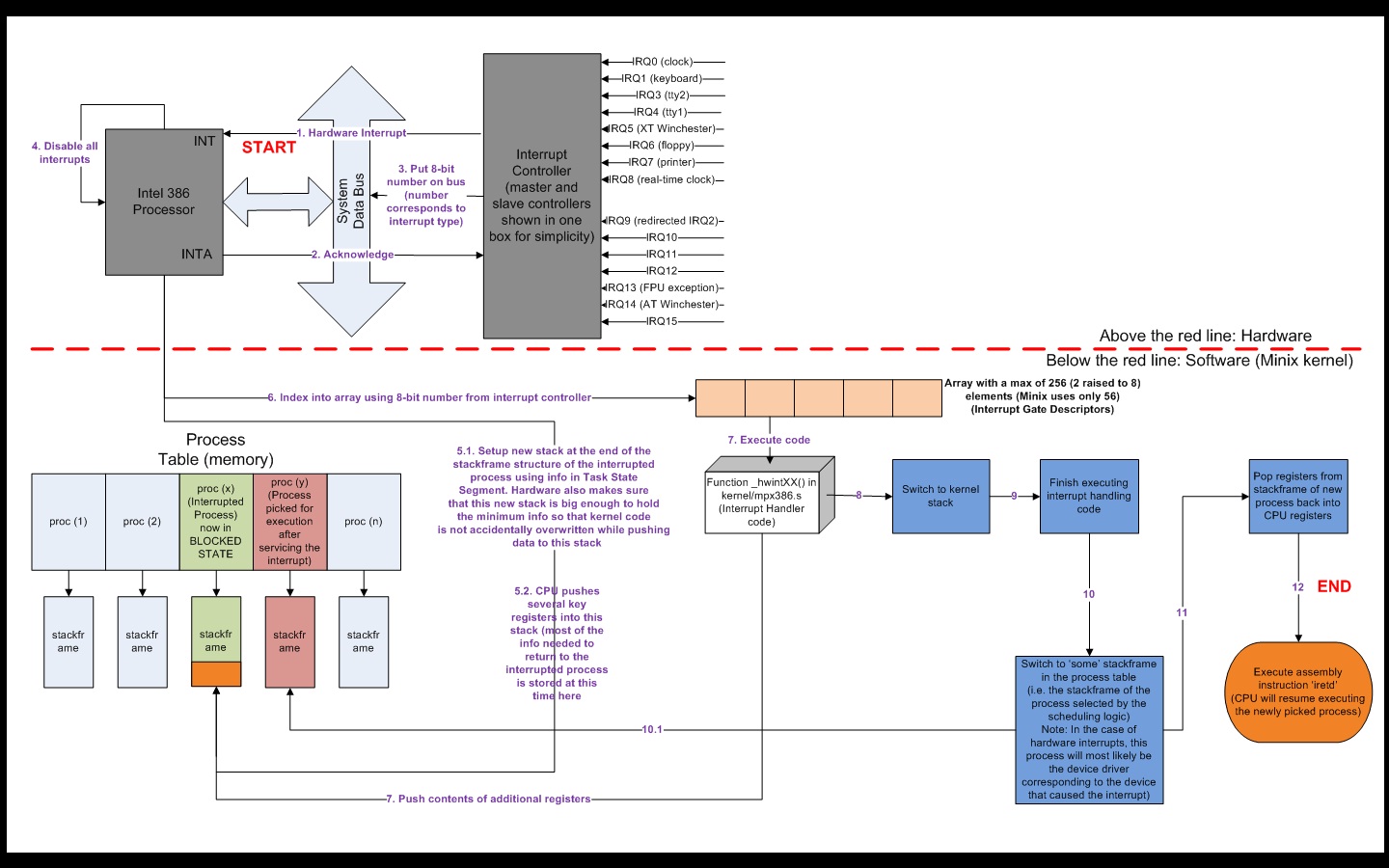
Interrupt vector is the location of near the bottom of memory associated with I/O device class

It contains the address of the interrupt service procedure.

Suppose user process 3 is running when a disk interrupt occurs.

1. The program counter, program status word, and possibly one or more registers are pushed onto the stack by the interrupt hardware.
2. The computer jumps to the address specified in the disk interrupt vector. That is all the hardware does.
3. From here, it’s software. The interrupt service procedure starts out by saving all the registers in the process table entry for the current process. Then the information deposited by the interrupt is removed from the stack, and the stack pointer is set to a temporary stack used by the process handler.

Refer to p.369~370



Interrupt service procedures are assembly language routine and then C procedure does the rest of work if needed.

1. Now we need to notify disk process message that interrupt occurred, using interprocess communication (in MINIX, via message)

2.5. OVERVIEW OF PROCESS IN MINIX

interprocess communication – message passing

2.5.1 The Internal Structure of MINIX

Refer to Fig. 2-26

1. process management
2. catching traps and interrupts, saving and restoring registers and general nuts and bolts of actually making the process abstraction provided to the higher layers work.
3. Handling the mechanics of messages
4. I/O processes (device drivers)

* layer 1, 2 are together called kernel.

1. memory manager and file system
2. MM carries out system calls involved with memory management, such as FORK, EXEC, and BRK.
3. FS carries out system calls, such as READ, MOUNT, and CHDIR

Operating systems do two things: manage resources and provide an extended machine by implementing system calls. In MINIX, the resource management is largely in the kernel (layer 1, 2), and system call interpretation is in layer 3. The file system has been designed as file “server”

1. user processes – shells, editors, compilers

2.5.2 Process Management in MINIX

all the user processes in the whole system are part of a single tree with init at the root. P.89 first half

2.5.3. Interprocess Communication in MINIX

By exchanging fixed-size messages. The size of the messages is determined by the size of a structure called message. On the 8088 it is 24 bytes.

Three primitives are provided for sending and receiving messages.

Send(dest, &message);

Receive(source, &message);

Send\_rec(src\_dst, &message); // to send a message and wait for a reply from the same process. The reply overwrites the original message. P.90 first phase

2.5.4 Process Scheduling in MINIX

multilevel queueing system with three levels, corresponding to layers 2, 3 and 4. With in each level, round robin is used. Task have the highest priority, the memory manager and file server are next and user processes are last.

Read 2.5.4 on p.90

2.6 IMPLEMENTATION OF PROCESSES IN MINIX

2.6.2. The Common Header Files

p.93 circle about ‘extern’

[callnr.h]

: describes system call number

[com.h]  
: definitions used in messages from MM and FS to the I/O tasks.

To distinguish them from process numbers, task numbers are negative.

[error.h]

: error messages that are returned to user programs in errno when a system call fails, as well as some internal errors

[type.h]

: It contains a number of key type definitons. For example, MAX(x, y)

[kernel/proc.h]

: It contains the process table.