CMPSCI 577 Operating Systems Design and Implementation **Spring 2020**

Lecture 8: 2/13/2020

Lecturer: ***Prashant Shenoy*** *Scribe:* ***Alexander Guerriero (Undergrad Senior)***

.1 Final Words on Scheduling

In Minix, the high level scheduling policy is implemented in user space. It still uses an MLFQ to implement it's scheduling policy. The kernel has a small default policy and does the work of running the scheduled process.

To implement this, the kernel exposes two kernel calls sys_schedule and sys_schedctl which allow the process manager to determine which process should be run next.

Now that the scheduler is in user space it allows the opportunity to implement multiple schedulers for different process groups (this is not currently implemented in Minix). This would allow for different schedulers per user, per device type, etc, and would allow for better utilization of the CPU and better load balancing.

.2 Minix IPC

Minix defines 7 different predefined message types in ipc.h. Each message type is a fixed length and can hold different parameter types. These messages can be sent using calls to send and received using calls to receive. There is also a call named sendrecv which can be used to perform a send and a receive in one call. The message structure is a union on the 7 predefined message types and the m_source field shows who sent the message and m_type shows the message type. Minix define marcos to make it easier to use the message struct.

Messages are sent using rendezvous communication (blocking) and the functions mini_send mini_recv, and mini_notify do the actualy work for sending. These are defined in proc.c.

.3 Interrupt Handling

At the hardware level an interrupt is a signal going from low voltage to high voltage to signal something to the CPU. When the CPU receives an interrupt, it will save the state of currently running program and index into the interrupt vector to find the interrupt handler and jump to it.

Because interrupts are a hardware feature they are architecture dependent and require architecture specific code.

Because Minix implements device drivers in user space interrupts are a little more complicated than this. In the interrupt vector is a stub for the interrupt handler since the real interrupt handler is in the device driver. The kernel will instead up call to the correct device driver. This incurs an overhead that does not happen in monolithic kernels.

.4 Process Management

A **process** encapsulates a running program and its execution context (pc, registers, virtual memory, open files, etc). It should be noted that a process and a program are not the same thing.

The operating system manages processes using a data structure called the **process table**. The process table holds the execution state and location for each process. When a new process is created a new entry is added in the process table. Because of the micro-kernel architecture for Minix, the process table is distributed across the process manager, the memory manager, and the file system.

A **thread** is a single sequential execution stream within a process. It is bound to a single process. Threads can be implemented exclusively in user space or with kernel awareness. Each provide their own pros and cons.

For a user space implementation of threads, the OS scheduler picks a process and the threading library within that process decides which thread will run.

Process creation can happen at a number of different times. Processes are created during system initialization, by already running programs which call the fork system call, when a user requests to create a new process, and by batch jobs.

The fork system call is used to create a child process. The parent can wait for the child or continue running. After the child can choose to call the exec system call to load a new program into memory.