【OS】Day7(2)

【Ch2】Process（3）

4.5 Data Structures

The OS is a program, and like any program, it has some key data structures that track various relevant pieces of information.

To track the state of each process, for example, the OS likely will keep some kind of process list for all processes that are ready and some additional information to track which process is currently running.

The OS must also track blocked processes; when an I/O even complets, the OS should make sure to wake the correct process and ready it to run again.

```
// the registers xv6 will save and restore
// to stop and subsequently restart a process
struct context {
  int eip;
  int esp;
  int ebx;
  int ecx;
  int edx;
  int esi;
  int edi;
  int ebp;
};

// the different states a process can be in
enum proc_state { UNUSED, EMBRYO, SLEEPING,
                  RUNNABLE, RUNNING, ZOMBIE };

// the information xv6 tracks about each process
// including its register context and state
struct proc {
  char *mem;                  // Start of process memory
  uint sz;                    // Size of process memory
  char *kstack;               // Bottom of kernel stack
                              // for this process
  enum proc_state state;      // Process state
  int pid;                    // Process ID
  struct proc *parent;        // Parent process
  void *chan;                 // If !zero, sleeping on chan
  int killed;                 // If !zero, has been killed
  struct file *ofile[NOFILE]; // Open files
  struct inode *cwd;          // Current directory
  struct context context;     // Switch here to run process
  struct trapframe *tf;       // Trap frame for the
                              // current interrupt
};
```

Figure 4.5: **The xv6 Proc Structure**

The register context will hold, for a stopped process, the contents of its registers. When a process is stopped, its registers will be saved to this memory location; by restoring these registers(i.e. placing their values back into the actual physical registers), the OS can resume running the process. This technique is known as a context switch.

A process could be place in a final state where it has exited but has not yet been cleaned up(in UNIX-based systems, this is called the zombie state).

This final state can be useful as it allows other processes(usually the parent that created the process) to examine the return code of the process and see if the just-finished process executed successfully(usually, programs return zero in UNIX-based systems when they have accomplished a task successfully, and non-zero otherwise).

Summary

ASIDE: KEY PROCESS TERMS

- The **process** is the major OS abstraction of a running program. At any point in time, the process can be described by its state: the contents of memory in its **address space**, the contents of CPU registers (including the **program counter** and **stack pointer**, among others), and information about I/O (such as open files which can be read or written).
- The **process API** consists of calls programs can make related to processes. Typically, this includes creation, destruction, and other useful calls.
- Processes exist in one of many different **process states**, including running, ready to run, and blocked. Different events (e.g., getting scheduled or descheduled, or waiting for an I/O to complete) transition a process from one of these states to the other.
- A **process list** contains information about all processes in the system. Each entry is found in what is sometimes called a **process control block** (**PCB**), which is really just a structure that contains information about a specific process.