Chapter 4 The Process Abstraction

Most fundamental abstraction provide to user of an OS: PROCESS

Def. Process: a running program

Goal: Run N programs at the same time even though only M CPUs, where N >> M.

Illusion: each program thinks it has its own isolated machine.

CPU: (time sharing) A …. | B …. | C … … … | D . | E … | A … |

OS can do this by virtualizing the CPU (time sharing)

- Users can run as many concurrent processes as they like
- Potential cost: performance
  - The more processes you run, the less CPU time each process gets

To implement virtualization of the CPU, it is common to provide:

- **Mechanisms**: Low-level machinery
  - Implements low-level methods or protocols for some functionality
  - Ex: context switch: OS's ability to stop one running process and start running another
- **Policy**: High-level intelligence
  - On top of mechanisms we use *policies* (algorithms) to make decisions
  - Ex: scheduling policy: which program should run next?
    - Typically based on historical information

General software design principle that enable **modularity**:
Separation between policy and mechanism.

The Process Abstraction

Machine / CPU state:

What can a program read and update when it is running?

- the machine's state is a process's memory / registers …

Memory:

- Instructions are in memory
- Data that the running program reads/writes is in memory
- Memory of a process is called its **address space**.

Registers:

- A process's machine state include these registers
- Because many instructions read/update the registers

Special Purpose Registers

See link and this and this do understand what a stack and frame pointers are

- Program Counter (PC)
- Stack Pointer (SP)
- Frame Pointer (FP)

I/O Information

- List of open files

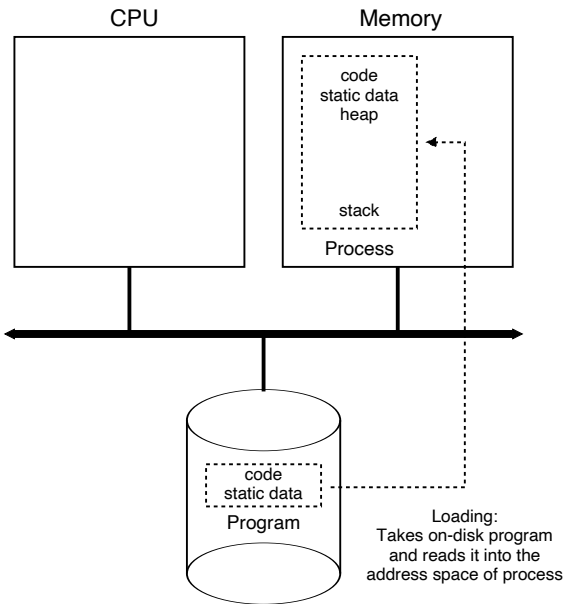
Process API

General API provide by modern OSes:

- Create
  - Type command into a shell (or terminal) or double-click an icon
- Destroy
  - Can kill running process forcefully (CTRL-C, kill <PID>, killall cpu, kill -9 <PID>)
- Wait
  - Can wait for process to stop
- Control
  - Suspend (CTRL-Z)
  - Resume a process (fg — foreground, bg — background)
- Status
  - Running time of process
  - What state is it in (ready, waiting, suspended ,…)

Process creation:

- Load into memory
- Point the PC @ the first instruction
- Go

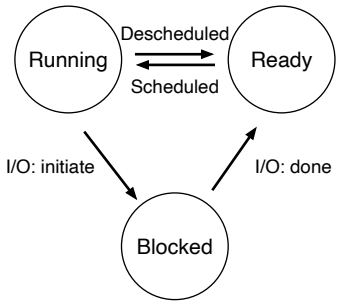


OS Does:

- Load code and static data into memory
- Allocate program's runtime stack
  - C programs use stack for local variables, function parameters, return addresses
  - Initialize stack with arguments (argc, argv)
- Allocate memory for program's heap
  - Used by program to ask for dynamically allocated data (malloc, free)
  - Used for data structures such as linked lists, hash tables, trees…
- I/O Initialization
  - Setup default file descriptors
  - Standard Input / Output / Error (stdin, stdout, stderr)
- Transfer control of CPU to newly created process
  - Special mechanisms: jumps to the main() function

Process States

Simplified view of process states:



- **Running**: process is running on a processor, executing instructions
- **Ready**: process is ready to run, but OS has chosen not to run it at this given moment
- **Blocked**: process has performed some operation that makes it not ready to run until some other event takes place.
  For example:
  - when a process initiates an I/O request to disk or network, it becomes blocked and thus some other process can use the processor.
  -

Time	Process_0	Process_1	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	Process_0 initiates I/O
4	Blocked	Running	Process_0 is blocked,
5	Blocked	Running	so Process_1 runs
6	Blocked	Running	
7	Ready	Running	I/O done
8	Ready	Running	Process_1 now done
9	Running	–	
10	Running	–	Process_0 now done

Figure 4.4: **Tracing Process State: CPU and I/O**

Time	Process_0	Process_1	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	
4	Running	Ready	Process_0 now done
5	–	Running	
6	–	Running	
7	–	Running	
8	–	Running	Process_1 now done

Figure 4.3: **Tracing Process State: CPU Only**

Data structures

OS keep track of

- Currently running process
- List of ready to run (runnable) processes (process list)
- Blocked processes

Register context: holds the content of a stopped proc's registers

- When proc stopped
  - Save register to this location (to main memory)
- When proc resume
  - Restore registers from this location (from main memory)

```
// 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 non-zero, sleeping on chan
  int killed;                 // If non-zero, have 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**

Summary of 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.

Questions:

- What is the role of scheduler in os?
- What is process list?
- What information is contained in the process control block PCB?