CS 330 - Operating Systems

Processes

07-08-2025

Process

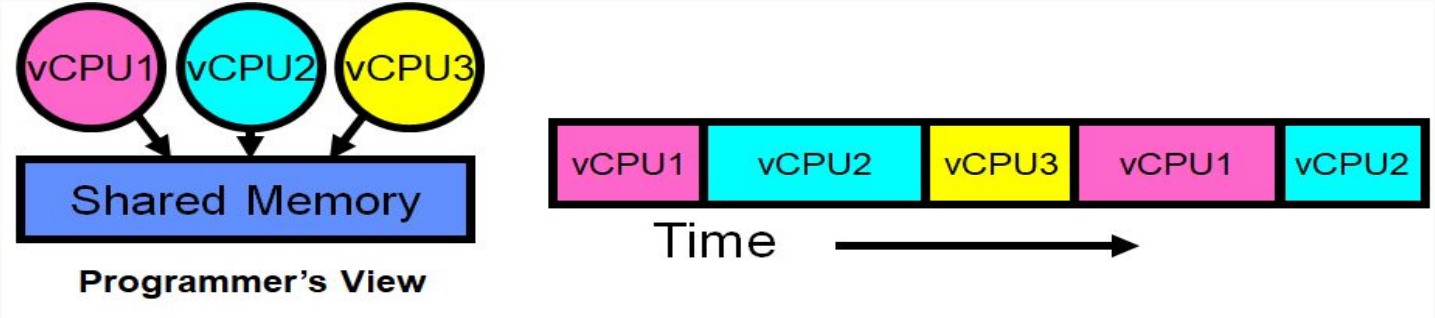
- Execution environment with restricted rights
  - Thread(s) + address space
  - Encapsulate one or more threads sharing process resources

Example

```
main() {
    ComputePI(“pi.txt”);
    PrintClassList(“classlist.txt”);
}
```

Threads

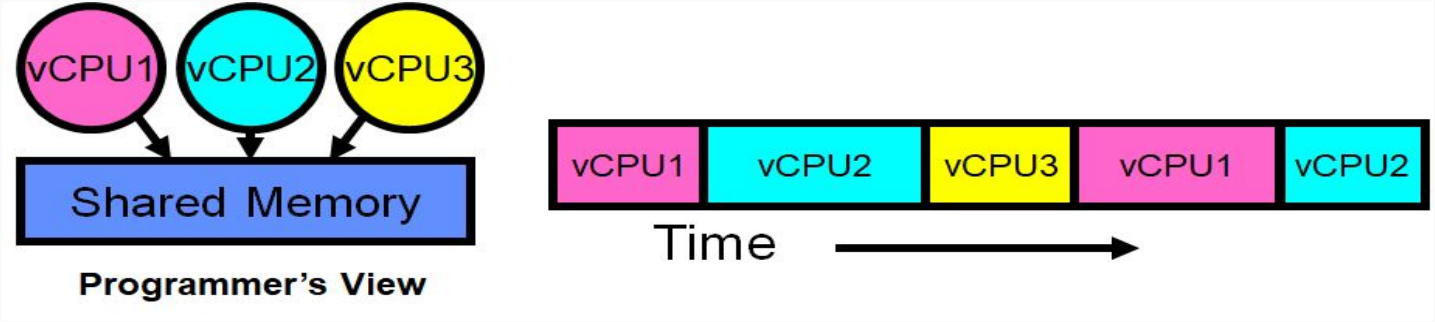
- Threads are virtual cores (multiplex over time)



- Where is the thread in the system?

Threads

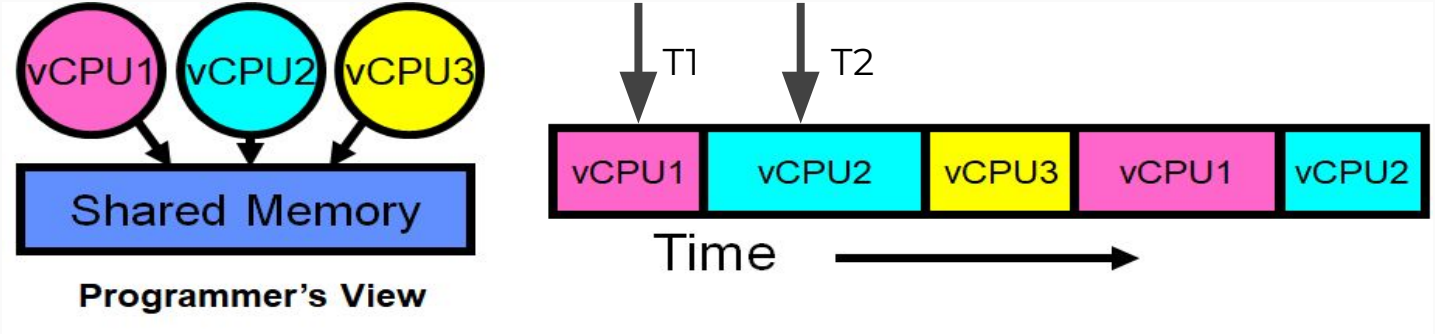
- Threads are virtual cores
  - Instances of the CPU at given point of time



- Thread is either on the core or in the memory

Threads

- Assume at T1 vCPU1 on real core; vCPU2 in memory
- Assume at T2 vCPU2 on real core; vCPU1 in memory



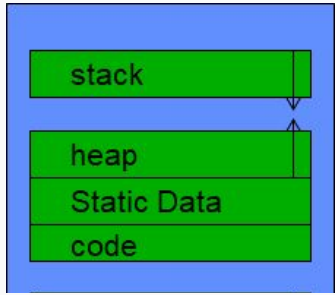
- What are the steps involved?

Threads

- Single execution context
  - Registers (state of data)
  - Program counter (spl. register -- contains next instruction)
  - Memory state
  - Stack (function call information)
- Context Switch
  - Thread control blocks (TCBs) save state

TCB

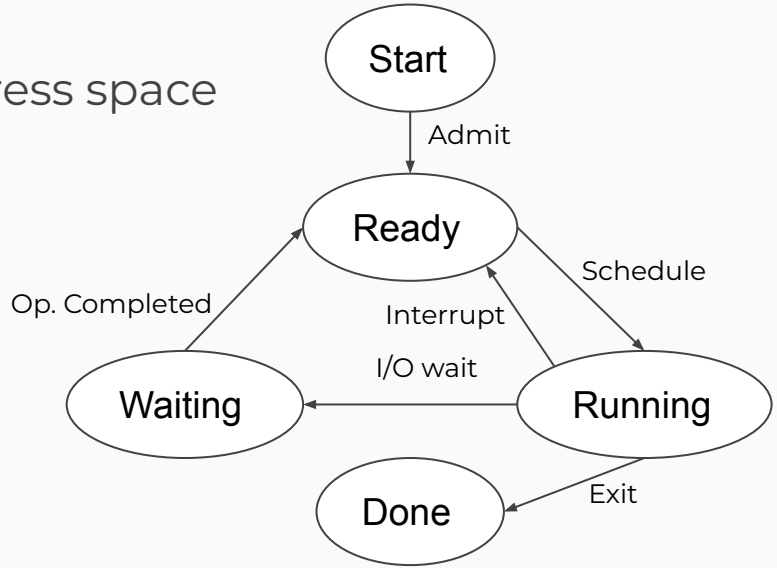
- Thread Control Block
  - Holds contents of registers for suspended thread
  - Thread Identifier: Unique id (tid) is assigned to every new thread
  - Stack pointer: Points to thread's stack in the process
  - Program counter:
    Points to the current program instruction of the thread
  - State of the thread (running, ready, waiting, start, done)
  - Thread's register values
  - Pointer to the Process control block (PCB) of the process that the thread lives on



- Where are the TCBs stored?

Process

- Execution environment with restricted rights
  - Thread(s) + address space
  - Encapsulate one or more threads sharing process resources
- Has similar state as threads
  - PC, SP, registers along with address space
- States



Process Control Block

```
// 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
};
```

Process APIs in UNIX Systems

- fork()
  - fork() system call is used to create a new process
- exec()
  - if you want to run a different program (not the one in current)
- wait()
  - parent to wait for a child process to finish what it has been doing
- kill()
  - kills the process specified by the identifier