Define process? 

Goal – provide illusion that each process has its own CPU.

Basic solution: Time Slicing

Process as an ADT

Any modern operating system will have the following as part of its API:

1. Creating a new process
2. Destroying a process (used when a process must be “killed” due to failures)
3. Waiting for a process to stop running
4. Miscellaneous Control, such as suspending a process
5. Status updates on a process

Cake Example

Program – recipe
Input – ingredients
CPU – mixer
Output – cake
Interrupt – Door bell
Multi-processing –:
Blocked ⋄ -- no eggs

Process States (consider transitions policy versus mechanism)
Let's trace two processes through this state diagram:

<table>
<thead>
<tr>
<th>Time</th>
<th>Process0 state</th>
<th>Process1 state</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Running</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Running</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Running</td>
<td></td>
<td>Process0 initiates I/O</td>
</tr>
<tr>
<td>4</td>
<td>BLK</td>
<td>Running</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>I/O done</td>
</tr>
<tr>
<td>7</td>
<td>Rdy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Running</td>
<td></td>
<td>P1 finish</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When did we have to make a decision? What is the impact?

**Process Creation**

- Load code and data from disk file into memory.
- Create (empty) call stack.
- Create and initialize process control block (PCB).
- Make process known to dispatcher.

(Fork Process Creation)

<table>
<thead>
<tr>
<th>PCB</th>
<th>PCB</th>
<th>PCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>Fork(); Code</td>
<td>Fork(); Code</td>
<td>Fork(); Code</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Stack</td>
<td>Stack</td>
<td>Stack</td>
</tr>
</tbody>
</table>

Before

Two (almost) identical copies

After

Wait, after the fork call the two are the same? That can't be right? What's different?

**How does the OS track important information?**
```c
/* typical server code */

main()
{
    int s, t;

    s = create_socket(PORT_NUMBER);
    if (s < 0)
    {
        perror("create");
        exit(1);
    }

    for(;;)
    {
        if ((t = wait_for_connection(s)) < 0)
        {
            if (errno == EINTR)
                continue;
            perror("wait_for_connection");
            exit(1);
        }

        switch(fork())
        {
            case -1:
                perror("fork"); /* can't start child process */
                close(s);
                close(t);
                exit(1);

            case 0: /* child */
                close(s);
                handle_request(t);
                exit(0);

            default: /* parent */
                close(t);
                continue;
        }
    }
}
```
What (state information) does the OS need to keep track of?

The xv6 Proc Structure

// the registers xv6 will save and restore
// to stop and subsequently resart a process
struct context {
    int cip;
    int csp;
    int ebx;
    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 *file[NFILE]; // Open files
    struct inode *cwd; // Current directory
    struct context context; // Switch here to run process
    struct trapframe *tf; // Trap frame for the
                // current interrupt
};