Unix Process States

Fig: 3.15 from OS design & Principles, Stalling 1998

Process Creation in Unix

- One Process can create another process:
  - the original process is called the parent
  - the new process is called child
  - the child is an identical copy of the parent (same code, same data) but has a new process ID.
  - the parent can either wait for the child to complete or continue execution in parallel with child.

- Useful function calls:
  - Fork():
    - in child process, fork() returns 0.
    - in parent process, fork() returns process ID of child.
  - Execv():
    - Child can overwrite its remaining programs with a new one and start a completely different program.
  - Wait(pid):
    - Parent, if desired can wait until child completes.
# Example of Unix Process Creation

```c
#include <sys/types.h>
#include <stdio.h>

int a = 6; /*global (external) variable*/

int main(void)
{
    int b; /*local variable*/
    pid_t mypid, childpid; /*process ids*/
    printf("Before fork..\n");
    childpid = fork();
    mypid = getpid();
    if(childpid == 0) /*child*/
        a++; b++;
    else /*parent*/
        wait(childpid);
    printf("After fork..\n");
    printf("me=%d, mychild=%d, a=%d, b=%d\n", mypid, childpid, a, b);
    exit(0);
}
```

Aegis: fork

Before fork...
After fork...

mypid=101, mychild=0, a=7, b=89
mypid=80, mychild=101, a=6, b=88

---

# Schedulers

- **Long-term Scheduler**
  - Selects jobs from spooled jobs and loads into memory.
  - Executes infrequently, maybe only when a process leaves system.
  - Controls degree of multiprogramming:
  - Goal: good mix of CPU and I/O bound processes.
  - Does not really exist in modern time sharing systems.

- **Medium-term Scheduler**
  - On time sharing system does some of the task of long term scheduler.
  - May swap processes in and out of memory temporarily.
  - Goal: balance load for better throughput.
Short-Term Scheduler

- Executes frequently (about 10 times per second)
- Runs whenever:
  - Process switches from running to blocked
  - Time slice runs out for a process (timer interrupt)
  - Any other interrupt occurs
  - Process is created or terminated
- Selects process from those that are ready to execute, allocates CPU to that process
- Goals:
  - minimize response time
  - maximize throughput
  - Efficient use of resources
  - minimize overhead (such as context switching)
  - Fairness

Context Switching

Stopping one process and starting another is called context switch:

- saving all hardware registers (PC, SP etc) or any other process state info in that the stopping process' PCB.
- Loading all the Hardware registers of the new process from the new process' PCB

It is an expensive operation:

- A time sharing system may do 100-1000 context switches per second.
Next Class

Concurrent Process