Process & Process Descriptor (PCB)

Contents of a descriptor maps directly to the Abstract Machine provided by the OS

- Static variables
- Dynamically allocated variables
- Runtime stack
- Code
- PC, status, exec time priority
- Interface provided by OS

国有资产
One Program / Multiple Instantiations

Note:
Each Process has its own descriptor
- text (shared), data...

Only one process active at a time
(context switching)
UNIX Parent and Child Processes

Shared Program Text

Parent process

Child processes

Shared Files & Resources
Thread (Child Process)

- Thread: light-weight process
  - OS maintains minimal internal state information

- Usually instantiated from a process

- Each thread has its OWN unique descriptor
  - Stack, Thread Status Word (TSW)

- SHARES with the parent process (and other threads)
  - Program text
  - Files & Resources
  - Parent process data segment
Thread ...

Each thread is sharing/executing the EXACT same code

Unique for each thread
Minimal info
=> Light-weight

Shared components
Only 1 copy of descriptor in OS
Process creation - fork()... example

```c
int pidValue;

pidValue = fork();  /* creates a child process */
If(pidValue == 0) {
    /* pidValue is ZERO for child, nonzero for parent */
    /* The child executes this code concurrently with Parent */
    childsPlay(..);  /* A locally-liked procedure */
    exit(0);  /* Terminate the child */
}
/* The Parent executes this code concurrently with the child */

wait(..);  /* Parent waits for Child’s to terminate */
```

UNIX process creation : fork() facility
Process creation – Unix fork()…

- Child/Parent code executed based on the pid value in “local” data space
  - For parent process, pid value returned is that of the *child* (non-zero)
  - For child process, pid value returned is 0

- pidvalue returned to parent process is non-Zero

- Therefore, fork() creates a new LW process

```
Initial process
```

```
Parent process (HW)

fork()

Child process (LW)
```
Process Creation – Unix exec()

- Turns LW process into autonomous HW process

- fork()
  - Creates new process

- exec()
  - Brings in new program to be executed by that process
  - New text, data, stack, resources, PSW, etc.
    - BUT using same (expanded) process descriptor entries

In effect, the “exec’ed” code overlays “exec’ing” code
Process creation – exec()... example

```c
int pid;
..
     /* Setup the argv array for the child */
..
if((pid = fork()) == 0) {       /* Create a child */
    /* The child process executes changes to its own program */
    execve( new_program.out, argv, 0 );
    /* Only return from an execve call if it fails */
    printf("Error in execve");
    exit(0);                     /* Terminate the child */
}
/* Parent executes this code */
..
wait(..);                     /* Parent waits for Child’s to terminate */
```

UNIX process creation: exec() facility