Processes

Lesson 09 # Operating Systems I
Viktor Iakovlev (Victor Yacovlev)

The Process is...

- An instance of some running program
- An isolated address space of memory in the system

Process Attributes

Memory-Related Attributes:

- Processor Registers
- Memory Translation Tables
- Private and Shared memory pages
- Files mapping
- Individual system calls stack

Process Attributes

<u>FS-Related Attributes:</u>

- Descriptors table
- Current Working Directory
  why there is no ‹cd› program?
- Корневой каталог
  `to be changed by superuser`

Process Attributes

Other Attributes:
- Environment Variables
- Limits
- Resource Counters
- User and Group Identifiers

How to get an Information

- Command `ps` - show process list
- Command `top` - resources usage
- File System `/proc`

Жизненный цикл процесса

- ***R***unning
- s***T***opped
- Suspended
  - ***S***uspended - can be terminated
  - ***D***isk Suspended - can not be terminated
- ***t***racing
- ***Z***ombie

State change examples

```
sleep(10);
// from R to S

read(0, buffer, sizeof(buffer));
// possible from R to S

read(fd, buffer, sizeof(buffer)); // in case of file
// possible from R to D

_exit(5);
// from R to Z

raise(SIGSTOP);
// from R to T
```

Idleness

```
while (1) {
    // do nothing - just waste CPU
}
```

```
while (1) {
    sched_yield(); // OK
}
```

Process Creation

```
pid_t resut = fork()
```

`Creates` **`a`** **`copy`** `of current process`

- `-1 == result` -- error
- ` 0 == result` -- for a child
- ` 0 <  result` -- for a parent, in that case `result` - is a Process ID for a newly created process

Example

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

int main() {
  pid_t result = fork();
  if (-1 == result) { perror("fork :-("); exit(1); }
  if ( 0 == result) { printf("I'm son!\n"); }
  else {
     printf("I'm parent!\n");
     int status;
     waitpid(result, &status, 0);
     printf("Child exited with status %d\n", status);
  }
}
```

Process Copy

- Memory, registers etc. - an exact copy (except `%eax`/`%rax`)

- Differs in child process:
  - Process ID [`getpid()`], Parent ID [`getppid()`]
  - Pending Signals
  - Timers
  - File locks

Process Copy Side-Effects

```c
int main() {
  printf("abrakadabra ");
  pid_t result = fork();
  if (0==result) {
    printf("I'm son\n");
  }
  else {
    printf("I'm parent\n");
  }
}
```

```
abrakadabra I'm son
abrakadabra I'm parent
```

When it is not possible to start a process

- `/proc/sys/kernel/pid_max` [32768]
  Maximum count of processes in the system

- `/proc/sys/kernel/threads-max` [91087]
  Maximum count of threads, so each process have at least one thread

```
shell>  :(){ :|:& };:
```

Disclaimer!

This code is too dangerous!

```
void fork_bomb() {
  pid_t p;
  do {
    p = fork();
  } while (-1 != p);
  while (1) sched_yeild();
}
```

Fork-Bomb demonstration

Process Tree

- The first has number `1` - ~~`init`~~ **`systemd`**
- All but ~~`init`~~ **`systemd`** have a parent
- Process 1 becames a parent of orphaned process in case of parent death
- Parents are noticed in case of child death

Process Termination

- System call `_exit(int)`
- Function `exit(int)`
- Operation `return INT` at `main`

```
printf("abrakadabra");
_exit(0)
```

```
printf("abrakadabra");
exit(0)
```

Process Termination

- Function `exit`:
  - calls all handlers registered by `atexit`
  - flushes I/O buffers
  - calls system call `_exit`

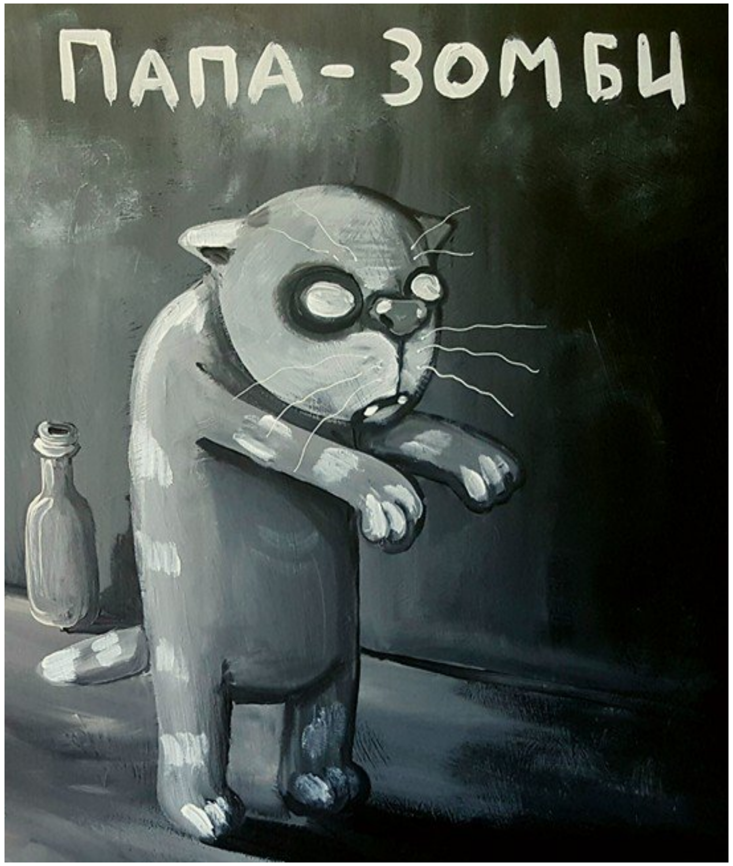
Exit Status

- Process might exit itself by calling system call `_exit(0<=code<=255)`
- Process might be externally terminated by someone using signals

```c
int status;
waitpid(child, &status, 0);
if (WIFEXITED(status)) {
   printf("Exit code: %d", WEXITSTATUS(status));
}
else if (WIFSIGNALED(status)) {
   printf("Terminated by %d signal", WTERMSIG(status));
}
```

Zombie (<defunc>) Processes

- Zombie is the last process state
- Zombies are deleted from process table by parent which calls `wait` or `waitpid`
- Zombies must be cleared to prevent fork-bomb effect



exec - change current process to another program

man 3 exec

```
int execl(const char *path, const char *arg, ..., /* 0 */)
int execlp(const char *path, const char *arg, ..., /* 0 */)
int execle(const char *path, const char *arg, ..., /* 0 */, char * envp[])

int execv(const char *path, char * const argv[])
int execvp(const char *path, char * const argv[])

#ifdef _GNU_SOURCE
int execvpe(const char *path, char * const argv[], char * const envp[])
#endif
```

Example

```c
int main() {
  pid_t  pid = fork();
  if (-1==pid) { perror("fork :-("); exit(1); }
  if (0==pid) {
    execlp("ls", "ls", "-l", NULL);
    perror("exec :-(");
    exit(2);
  }
  else {
    waitpid(pid, NULL, 0);
  }
}
```

Example

```c
int main() {
  pid_t  pid = fork();
  if (-1==pid) { perror("fork :-("); exit(1); }
  if (0==pid) {

    // a place to make additional process
    // setup: environment and others

    execlp("ls", "ls", "-l", NULL);
    perror("exec :-(");
    exit(2);
  }
  else {
    waitpid(pid, NULL, 0);
  }
}
```

Example

```
int main() {
  pid_t  pid = fork();
  if (-1==pid) { perror("fork :-("); exit(1); }
  if (0==pid) {
    chdir("/usr/bin");
    int fd = open("/tmp/out.txt",
                  O_WRONLY|O_CREAT|O_TRUNC, 0644);
    dup2(fd, 1); close(fd);
    execlp("ls", "ls", "-l", NULL);
    perror("exec :-(");
    exit(2);
  }
  else {
    waitpid(pid, NULL, 0);
  }
}
```