Unix-like Operating Systems

Processes and threads. Signals.

Jan Trdlička

Czech Technical University in Prague, Faculty of Information Technology
Department of Computer Systems

Content

Process

- Instance of running program/application.
- Entity within which resources are allocated (memory, threads, open files, locks, semaphores, sockets,...).
- By default, each process has a single computing thread („main“ thread).
- Kernel of OS keeps a range of data structures for each process
  - security (identity of process),
  - memory management (information for translation of logical address,...),
  - FS management (file descriptor table,...),
  - CPU management (information for thread scheduling,...),
  - ...
- When a new process is created, one part of data structures is inherited from the parent process (file descriptor table,...) and the other part is set to new values specific to the new process (process number, ...).

Process

- Each process is identified by unique process identifier (PID) used by kernel.
- Each process knows its parent under the parent process identifier (PPID).
- New process is created by system call:
  - `fork()`
    - Function creates a new process.
    - The address space of the new process (child process) is an exact copy of the address space of the calling process (parent process).
    - Child process has some new properties (`PID`, `PPID`,...) and other properties are inherited (e.g. `EUID`, `EGID`,...) from parent process.
  - `exec()`
    - Function replaces the current process image with a new process.

Processes – example of creation

```
int main (void)
{ ...
   pid = fork();

   switch (pid) {

      case -1: /* error */
         perror ("error in fork()");
         exit (1);

      case 0:  /* child process*/
         printf ("PID of child: %d\n", (int) getpid ());
         execlp("sleep", "sleep", "30", (char *) NULL);
         perror ("error in execlp()");
         exit (1);

      default: /* parent process*/
         printf ("PID of parent : %d\n", (int) getpid ());
         wait(&status);
   };
   ...
}
```

Thread

- Computing entity that is allocated processor.
- Threads created within a process share the majority of the resources allocated in the process.
- Kernel of OS keeps a range of data structures for each thread
  - stack,
  - context switching (program counter, the current value registers,...),
  - CPU management (information for thread scheduling,...)

  ...
- Process with one thread (default) can use only one CPU at a given time.
- New thread can be created by library function, e.g. `pthread_create()`.

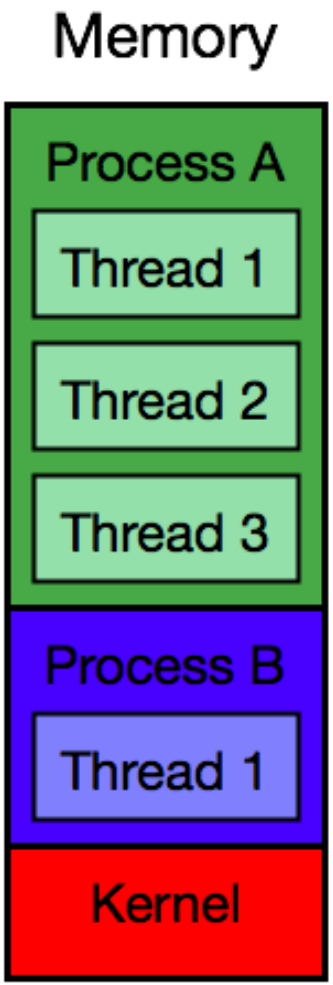
Thread – example of creation

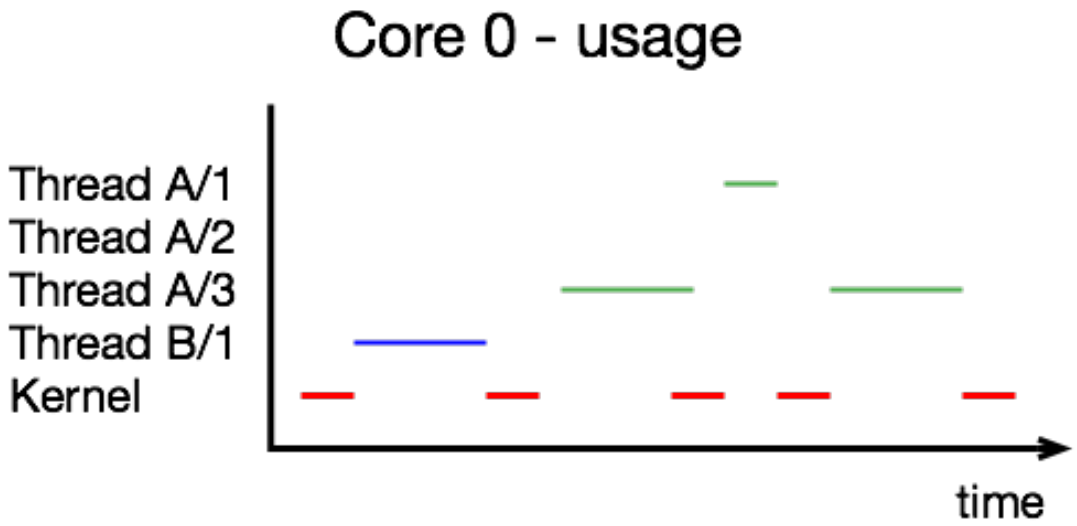
```
...
void *thread_code(void *threadid)
{
   printf("ID of thread: %ld\n", (long int) threadid);
   sleep(60);
   pthread_exit(NULL);
}
...
int main(void)
{
   pthread_t threads[NUM_THREADS];
   int rc, i;

   for ( i = 0; i < NUM_THREADS; i++ )
   {
     /* Create new thread */
     rc=pthread_create(&threads[i], NULL, thread_code, (void *) i);

     if (rc) { perror("error in pthread_create()"); exit(1); }
   }
...
}
```

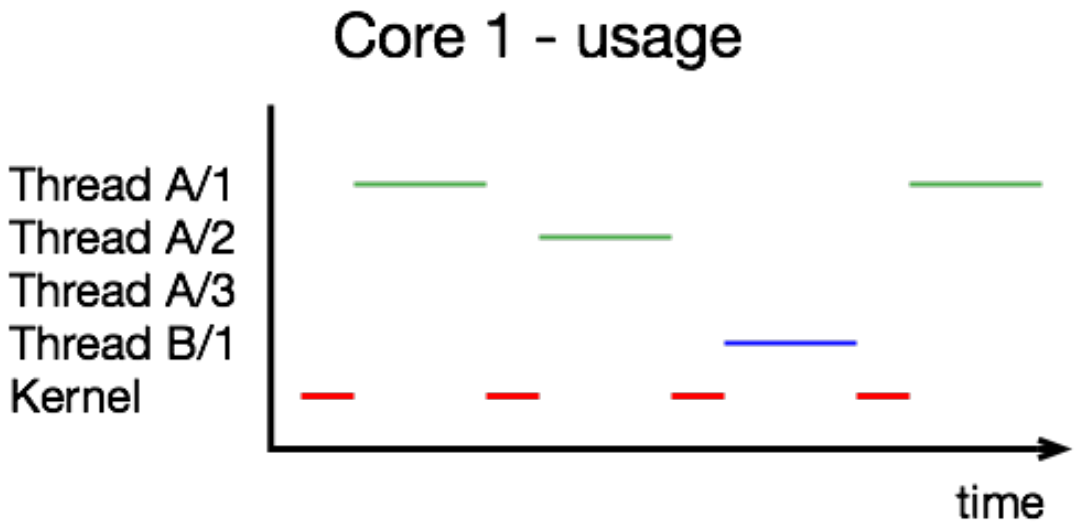
Context switching

- The process of storing the context of thread, so that it can be restored and execution resumed from the same point later.
- This enables multiple threads/processes to share available CPUs.
- Kernel determines (schedules) when and who gets the CPU.
- Kernel uses several information to make scheduling decision
  - Thread priority (e.g. 0 – 169 in Solaris).
  - Scheduling algorithm (e.g. TS, IA, FSS, FX, SYS, RT in Solaris).
  - Tread state, thread behaviour in history, ...
- Thread can use CPU only during some time quantum.
- Size of time quantum can be different for different threads and can vary in time (depends on Unix implementation).

Context switching

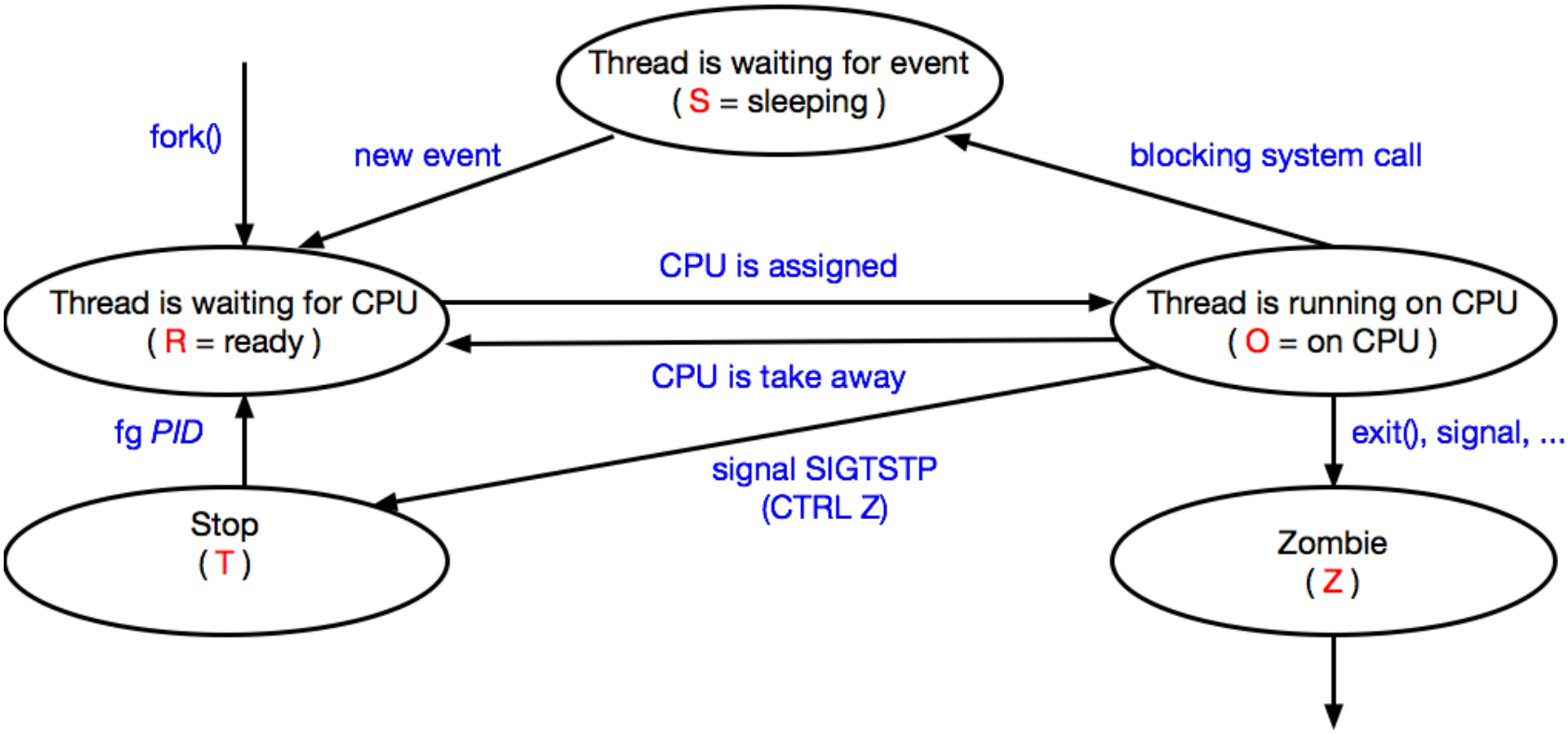






Thread states

- The thread state defines the phase in which the thead is located.
- Thread state diagram



Process information

- `ps`
  - Prints information about processes that have the same effective user ID and the same controlling terminal as the invoker.
- `ps -e`
  - Lists information about every running process.
- `ps -f` or item `ps -l`

S	state(O, S, R, Z, T)
PID, PPID	process ID, parent process ID
PRI	priority
NI	NICE value
STIME	starting time
TIME	cumulative execution time
TTY	controlling terminal
CMD	command name

Process information

- `ps -o` *format*
  - Option `-o` allows the output format to be specified under user control.
  - *format* specification must be a list of names:
    - `user ruser group rgroup uid ruid gid rgid pid ppid pgid sid pri nice class time etime stime s c lwp` ...
- `ps -Le`
  - Prints information about each thread (light weight process).
- `ps -U` *user_list*
  - Prints information about processes whose real user name or real ID is in the *user_list*.
- `ps -u` *user_list*
  - Prints information about processes whose effective user name or effective ID is in the *user_list*.

Process information

- `pgrep [ -lvx ] [ pattern ] [ -u euid... ]`
  - Reports the process IDs of the processes whose attributes match the criteria specified on the command line.
  - Useful options
    - `-l` ... prints the process name along with the process ID of each matching process.
    - `-v` ... inverse the sense of the matching.
    - `-x` ... only match processes whose names exactly match the *pattern*.
    - `-U` *ruid* ... only match processes whose real user ID is listed.
    - `-u` *euid* ... only match processes whose effective user ID is listed.

Process information

- `top`, `htop` or `prstat` (Solaris)
  - Iteratively examines all active processes on the system and reports statistics based on the selected output mode and sort order
- `pstree` or `ptree` (Solaris)
  - Prints the process trees containing the specified PIDs or users, with child processes indented from their respective parent processes

```
pstree -p
systemd(1)-+-ModemManager(974)-+-{ModemManager}(1084)
           |                   `-{ModemManager}(1093)
           |-Thunar(46111)
           |-VGAuthService(830)
           |-accounts-daemon(2651)-+-{accounts-daemon}(2652)
           |                       `-{accounts-daemon}(2654)
           |-agetty(3737)
           ...
```

Process priority

- Thread priority can be decreased (root can also increase) by command

  `nice -`*priority* *program*

  - Priority is integer number 1-19.
  - Higher number = lower priority.
  - Negative number = increasing of priority (only root).
  - In Solaris: better command for priority modification is `priocntl`.

Signals

- Limited form of inter-process communication used in Unix.
- Kernel interrupts the process's normal flow of execution, when signal is sent to a process, and starts signal handler.
- Signal can be send by
  - kernel
    - arithmetic exception, segmentation fault...,
    - terminal driver (e.g. key sequence CTRL C, CTRL, ...),
  - other process (e.g. command kill or function `kill()`).
- Signal is identified by name or number.
- Reaction to signal: none, termination, termination+core, stop, continue.
- How to print list of signals
  - Command: `kill -l`
  - In Unix manual: `man signal.7` (Linux)
- How to send signal
  - Command: `kill` *-signal PID*
  - Command: `pkill` *-signal* `[ -vx] [` *pattern* `-u` *ruid* `...]`

Useful signals

Signal	Meaning
SIGINT (2)	Interrupt from keyboard (`CTRL C`)
SIGQUIT (3)	Quit from keyboard (`CTRL \`)
SIGTSTP (24)	Stop typed at terminal (`CTRL Z`)
SIGTERM (15)	Termination signal (default)
SIGKILL (9)	Kill signal
SIGHUP (1)	Hangup detected on controlling terminal or death of controlling process

Signal trapping

- Default signal reaction are set during process startup.
- Process can modify signal reaction (except signals SIGKILL and SIGSTOP).
- We can also modify signal reaction by command `trap`.
  - Signal reaction setup

```
trap 'echo SIGINT' SIGINT
```

  - Print definition of signal reaction

```
trap
```

  - Setup ignoring

```
trap ' ' SIGINT
```

  - Setup default signal reaction

```
trap - SIGINT
```

Command execution

- In foreground

```
~> sleep 1000
```

  - Command (no built-in command) is executed like new process.
  - Standard input and outputs are assigned to terminal.
  - Shell waits for command termination.

- In background

```
~> sleep 1000 &
```

  - Command (no built-in command) is executed like new process.
  - Only standard outputs are assigned to terminal (if the command try to read from stdin the process is stopped).
  - Shell doesn't wait for command termination.

Command execution

- Immune to hang-ups

```
~> nohup sleep 1000 &
```

  - Process is running even after shell termination.
  - Standard outputs are redirected to file `nohup.out`.

- In given time

```
~> at 12:35 -f script_file
```

  - Command can be started at a given time by commands `at` or `crontab`.
  - System process cron executes the command under user identity at a given time.
  - Standard outputs are sent by email.