Lecture 7

Processes and threads.

Signals.

Department of Computer Systems FIT, Czech Technical University in Prague

Process

- Process is **running program**.
- 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)
  - function returns
    - -1 (error) or child PID in parent process
    - 0 in child 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

Example

```
int main (void)
{ ...
  pid = fork();
  switch (pid) {
  case -1: /* doslo k chybe */
    perror ("chyba ve funkci fork()");
    exit (1);
  case 0: /* kod provadeny v potomkovi */
    printf ("PID procesu potomka: %d\n", (int) getpid ());
    execlp("sleep", "sleep", "30", (char *) NULL);
    perror ("chyba ve funkci execlp()");
    exit (1);
  default: /* program provadeny v rodici */
    printf ("PID procesu rodice : %d\n", (int) getpid ());
    wait(&status);
  };
  ... }
```

Thread

- Thread is **running subprogram** in process/kernel environment.
- **Process containing *n* threads** can be executed **concurrently on *n* CPU's**.
- Process with one thread (default) can use only one CPU at a given time.
- **Thread creation is faster** then process creation.
- New thread can be created by library function, e.g. **`pthread_create(`**.

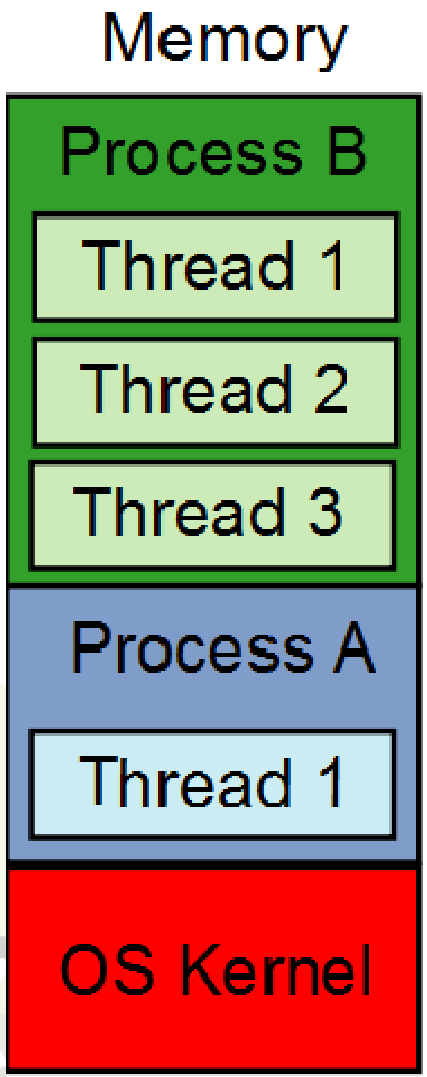
Example

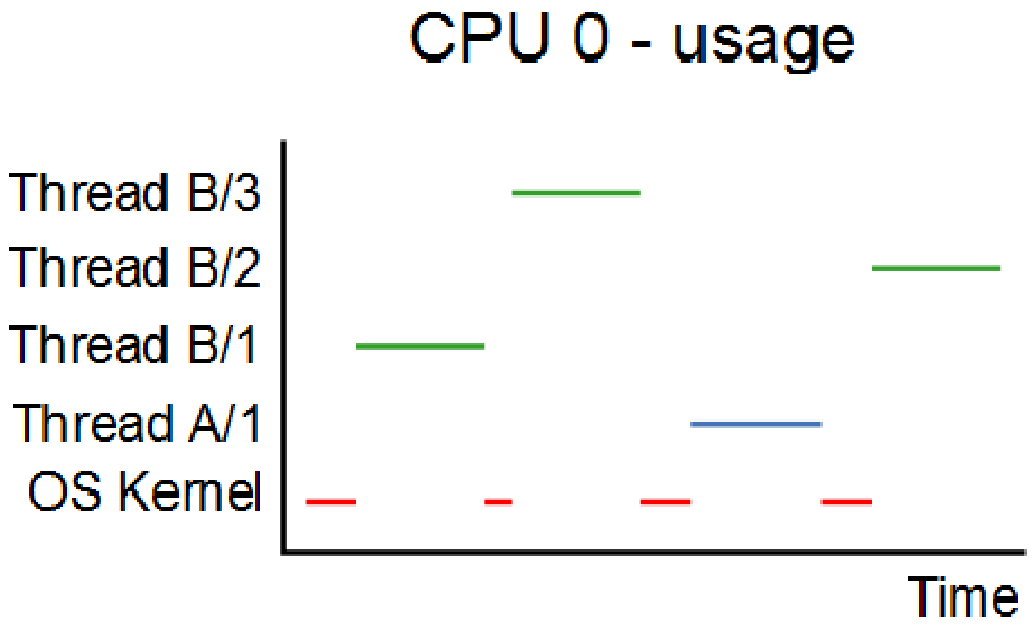
```
...
void *kod_vlakna(void *threadid)
{ printf("ID vlakna: %ld\n", (long int) threadid);
  sleep(60);
  pthread_exit(NULL);
}
...
int main(void)
{ pthread_t threads[NUM_THREADS];
  int rc, i;
  for(i=0; i<NUM_THREADS; i++){
    rc = pthread_create(&threads[i], NULL, kod_vlakna, (void *) i);
    if (rc){ perror("Chyba ve funkci pthread_create()"); exit(1); }
  }
  /* pthread_exit(NULL); */
  /* pthread_join(threads[i],&data); */
...}
```

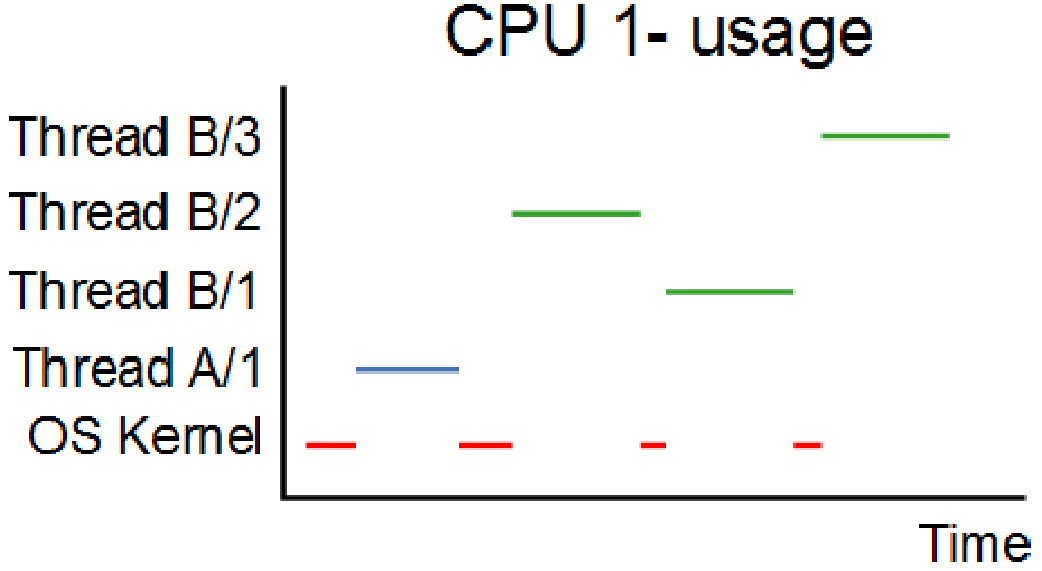
Context switching

- Computing process of storing and restoring state (context) of a CPU so that execution can be resumed from the same point at a later time
- This enables multiple threads/processes to share available CPUs.
- **Kernel determines when and who gets the CPU**.
- **Kernel uses several information to make decision**
  - Process/thread priority (e.g. 0 – 169 in Solaris)
  - Priority class (e.g. TS, IA, FSS, FX, SYS, RT in Solaris)
  - Tread state, thread behavior in history, ...
- **Thread/process priority** can be fixed or dynamic.
- Thread can use CPU only during some **time quantum.**
- **Size of time quantum** can be different for different threads/processes and can vary in time (depends on Unix implementation).

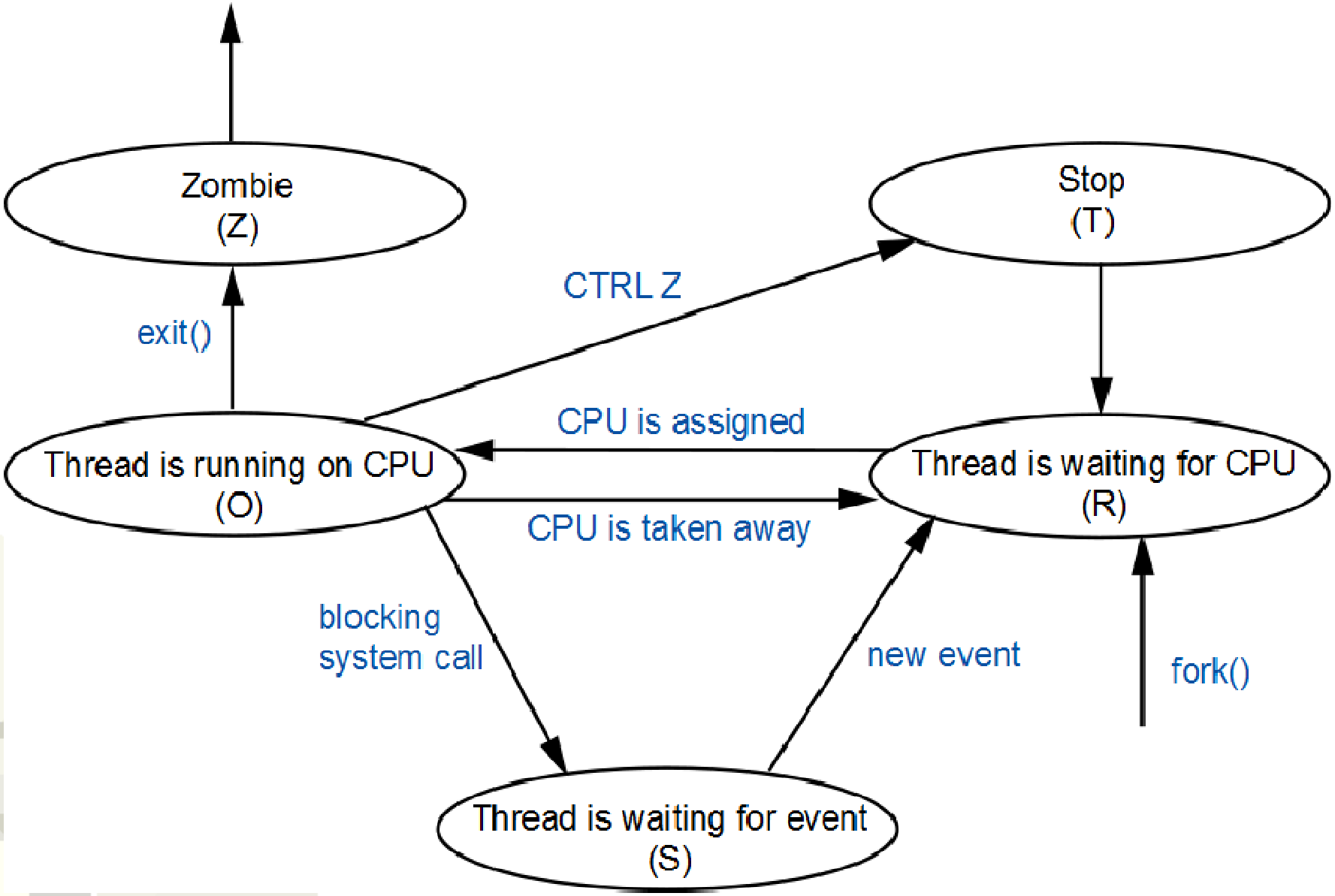
Context switching







Thread states



Process/thread listing

`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 **`ps -l`**

- Prints more details about running processes :
  - 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/thread listing

`ps -o format`

- 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)

Process/thread listing

`pgrep [–lvx] [pattern] [-u users...]`

- reports the process IDs of the processes whose attributes match the criteria specified on the command line
- -l Prints the process name along with the process ID of each matching process.
- –v Reverses the sense of the matching.

Process/thread listing

`prstat` or **`top`**

- iteratively examines all active processes on the system and reports statistics based on the selected output mode and sort order

`ptree [-a] [ pid ] [ user ]` in Solaris

`pstree [-a] [ pid ] [ user ]` in Linux

- prints the process trees containing the specified pids or users, with child processes indented from their respective parent processes

Process priority

- **Process priority can be decreased (root can also increase) by command**:

```
nice –priority program

nice –n priority program
```

- where **`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`**

How to write a correct script?

- What is the difference in the following loops?

```
while [ -f $1 ] ; do :          ; done; echo " $1 is removed "
```

```
while [ -f $1 ] ; do sleep 2 ; done; echo " $1 is removed "
```

- Which loop will determine earlier that the file is removed if the loop is executed concurrently 100x (every time with different file name $1)?

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.
- **Signal can be send by**
  - kernel (e.g. arithmetic exception, segmentation fault...)
  - terminal driver (e.g. key sequence CTRL C, CTRL \ , …)
  - jiným procesem (např. příkaz nebo funkce kill)
- Signal is identified by **name** and **number**.
- Reaction to signal: none, exit, exit+core
- **List of signals:**
  - Command **`kill –l`**
  - In Unix manual **`man –s 3HEAD signal`** (Solaris)
- **How to send signal:**

```
kill –signal PID
pkill –signal [ -vx] [ pattern –u users ...]
```

Signals and terminal driver

- **Some signal can be sent by sequence of keys:**
- (see `stty -a` or `man stty`)

Key sequences	Meaning	Signal
CTRL C	Interrupt process	2 SIGINT
CRTL \	Quit process	3 SIGQUIT
CTRL Z	Suspend process (not in sh). **Not process termination!!!**	24 SIGTSTP

Important signals

15 SIGTERM (TERMinate)

9 SIGKILL (KILL)

2 SIGINT (INTerrupt) and **3 SIGQUIT (QUIT)**

1 SIGHUP (HangUP)

- Parent process sends this signal to its child processes during termination
- If the child process must continue after parent process termination you must start children process by command

```
nohup příkaz &
```

Reaction to signal

- **Default signal reaction** are set during process startup.
- Process can **modify signal reaction** by system call **`trap()`** (except signals KILL and STOP).

- We can also modify signal reaction by command **`trap`**.
  - **Signal reaction setup**

    ```
    trap ´commands´ signals
    ```

  - **Print definition of signal reaction:**

    ```
    trap
    ```

  - **Setup ignoring:**

    ```
    trap ´ ´ signals
    ```

  - **Setup default signal reaction:**

    ```
    trap - signals
    ```

Command execution

- **In foreground**

  `$ command`

  - 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**

  `$ command &`

  - 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 it is stoped)
  - Shell doesn't wait for command termination.

Command execution

- **Immune to hang-ups**

```
$ nohup command &
```

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

- **In given time**
  - 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.

Job control

- Properties of shell (except /bin/sh)

- **Every process which is running in given shell is assigned job identifier (JID) in this shell.**

- In the shell we can use the following commands

  `jobs` list all running process inside this shell

  `fg [%JID]` move process to foreground

  `bg [%JID]` move process to foreground

  `kill –signál [%JID]` send signal to process

- If JID is not specified, the last process is used.

Useful commands

- **Elapsed time report**

  `timex command`

- **System calls used by command**

  `truss command`