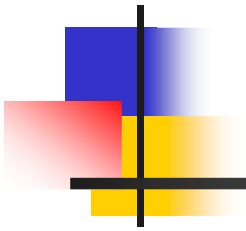
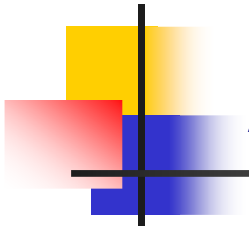
Linux Kernel



PROCESS



Linux Kernel : Process

- **Process** : a running program.

  Individual processes exist independently alongside each other and cannot affect each other directly. Each process's own area of memory is protected against modification by other processes.

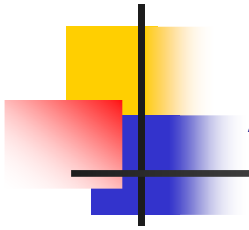
- **Thread** : a group of processes.

  Each thread runs within the same address space as the father process.

Processes and Threads

- Linux uses the same internal representation for processes and threads; a thread is simply a new process that happens to share the same address space as its parent.
- A distinction is only made when a new thread is created by the **clone** system call.
  - **fork** creates a new process with its own entirely new process context
  - **clone** creates a new process with its own identity, but that is allowed to share the data structures of its parent
- Using **clone** gives an application fine-grained control over exactly what is shared between two threads.

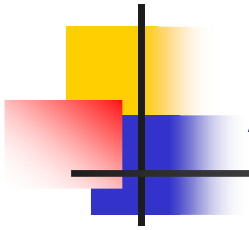
Flag	Meaning when set	Meaning when cleared
CLONE_VM	Create a new thread	Create a new process
CLONE_FS	Share umask, root, and working dirs	Do not share them
CLONE_FILES	Share the file descriptors	Copy the file descriptors
CLONE_SIGHAND	Share the signal handler table	Copy the table
CLONE_PID	New thread gets old PID	New thread gets own PID



Linux Kernel : Process

Process states

- **Ready** : the process is competing for the processor or could be executed, but another process is currently being executed.
- **Execution** : the process is active or running or being executed by processor.
- **Suspend** : the process is waiting for an external event.

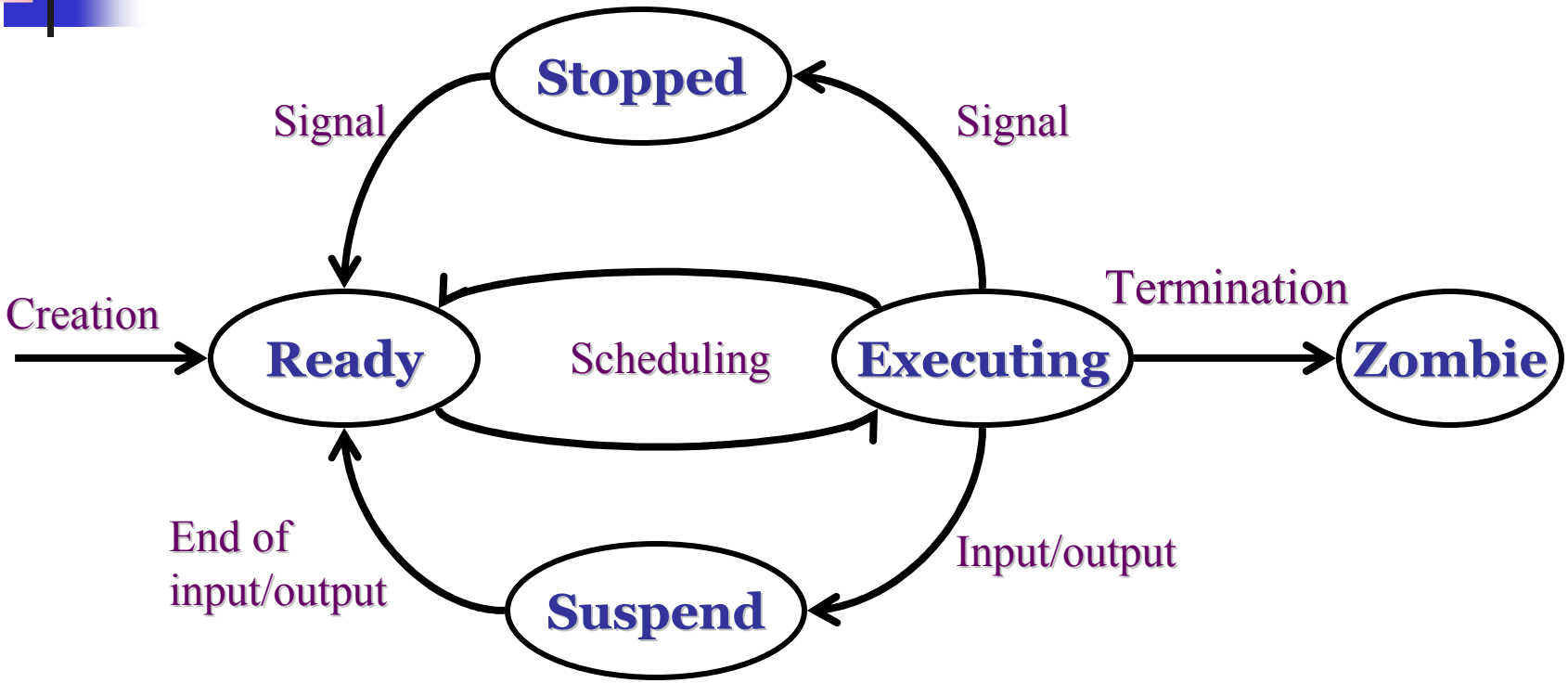


Linux Kernel : Process

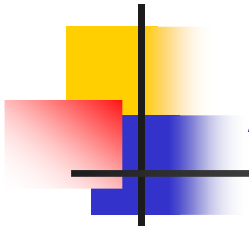
Process states *(cont)*

- **Stopped** : the process has been suspended by an external process.
- **Zombie** : the process has finished executed, but it is still references in the system.

Linux Kernel : Process



State diagram of a process.

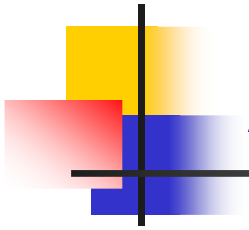


Linux Kernel : Process

Task structure

The description of the characteristics of a process is given in the structure **task_struct**, which is defined in **include/linux/sched.h** within Linux source code **(/usr/src/linux) .**

```
struct task_struct
{
    ...
    ...
};
```



Linux Kernel : Process

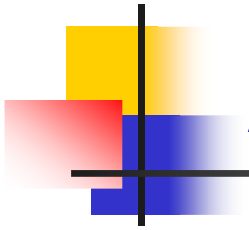
Task structure *(cont)*

- Process status

volatile long state;

The **state** variable contains a code for the current state of the process. List of state constants :

- **TASK_RUNNING**
- **TASK_INTERRUPTIBLE**
- **TASK_UNINTERRUPTIBLE**
- **TASK_STOPED**
- **TASK_ZOMBIE**
- **TASK_SWAPING**

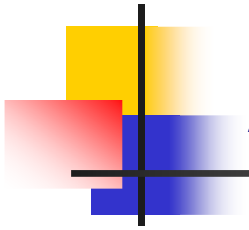


Linux Kernel : Process

Task structure *(cont)*

```
long counter;
long priority;
```

The **counter** variable holds the time in tick for which the process still run before mandatory scheduling action is carried out. The **priority** holds static priority of a process



Linux Kernel : Process

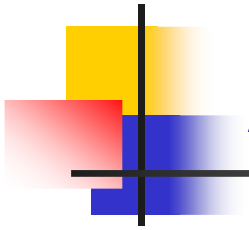
Task structure *(cont)*

```
unsigned long flags;
int errno;
int debugreg[8];
```

The **flags** contains the combination of system status flags **PF_ALIGNWARN, PF_PTRACED, PF_TRACESYS, PF_STARTING** and **PF_EXITING**.

```
struct exec_domain *exec_domain;
```

A description of which Linux is to be emulated for each process.



Linux Kernel : Process

Task structure *(cont)*

- Process identification

```
pid_t pid, pgrp, session;
int leader;
```

Every process has its own process ID, and is assigned to a process group and session. Every session has a leader process.

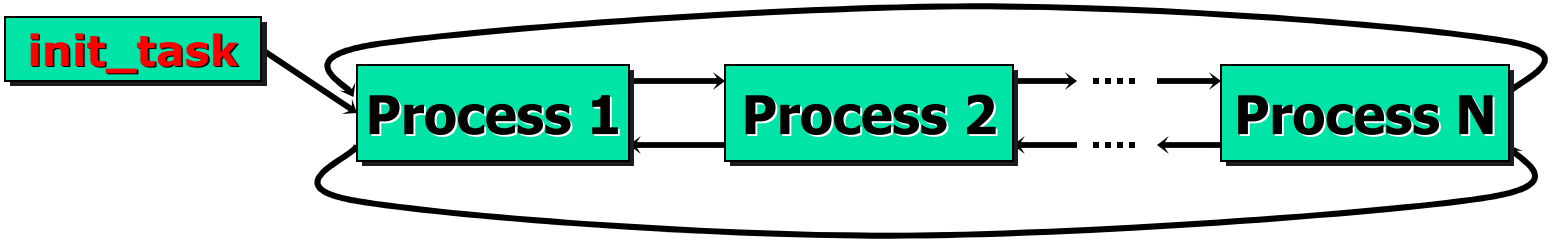
Linux Kernel : Process

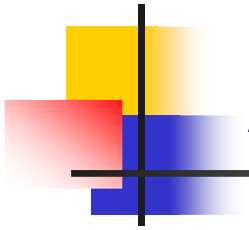
Task structure *(cont)*

- Process relationship

```
struct task_struct *next_task;
struct task_struct *prev_task;
```

All processes are entered in a doubly linked list which the start and end of this list are held in the global variable **init_task**.





Linux Kernel : Process

Task structure *(cont)*

- Process relationship *(cont)*

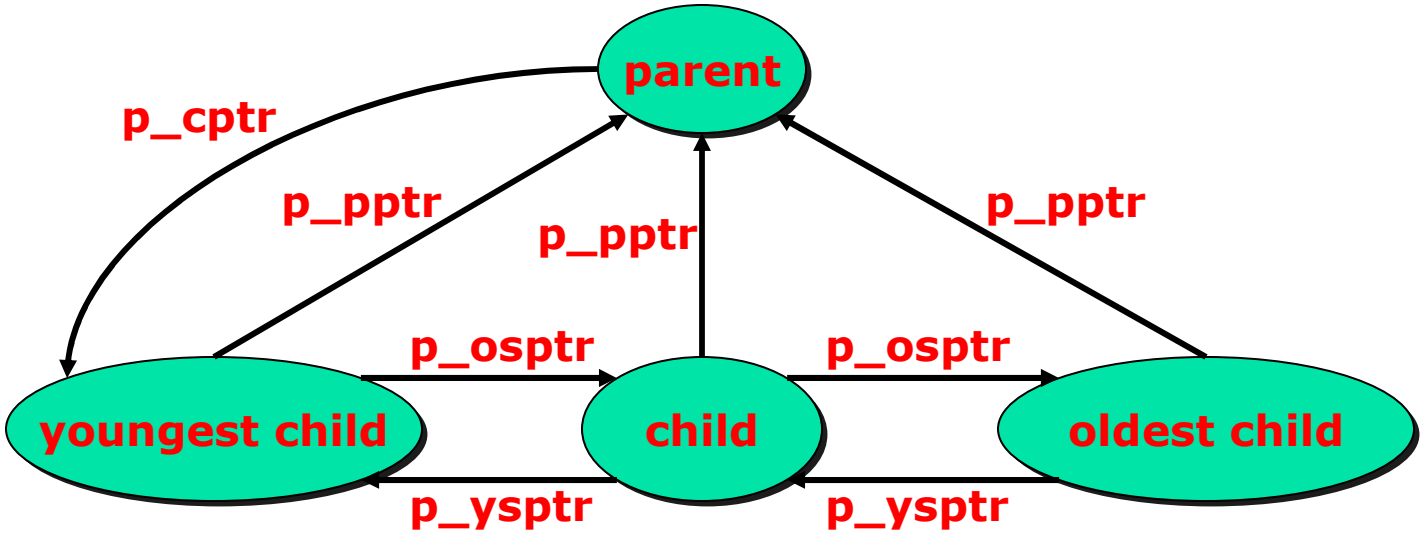
```
struct task_struct *p_opptr; /* original parent */
struct task_struct *p_pptr;   /* parent */
struct task_struct *p_cptr;   /* youngest child */
struct task_struct *p_ysptr; /* younger sibling */
struct task_struct *p_osptr; /* older sibling */
```

All processes have the 'family relationships' which has created it using the system call ***fork()***.

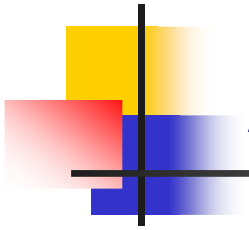
Linux Kernel : Process

Task structure *(cont)*

- Process relationship *(cont)*



The 'family relationships' of processes.



Linux Kernel : Process

Task structure *(cont)*

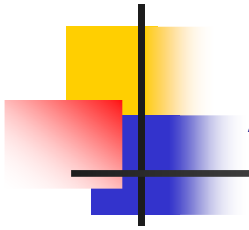
- Memory management

Each process holds their own memory substructure.

```
struct mm_struct *mm;
```

The component of memory structure:

```
struct mm_struct{
   unsigned long start_code, end_code, start_data, end_data;
   unsigned long start_brk, brk;
   unsigned long start_stack, start_mmap;
   unsigned long arg_start, arg_end, env_start, env_end;
};
```



Linux Kernel : Process

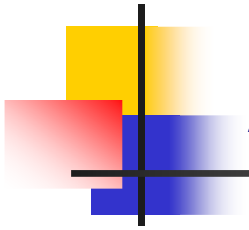
Task structure *(cont)*

- Process credentials

To handle access control, every process has a user ID, group ID and others.

```
uid_t uid, euid, suid, fsuid;
gid_t gid, egid, sgid, fsgid;
gid_t groups[NGROUPS];
```

These are inherited by the child process from the parent process. The **fsuid** and **fsgid** are used for actual access control to the file system.



Linux Kernel : Process

Task structure *(cont)*

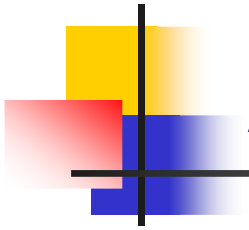
- File information

The file system structure is pointed from **fs** variable.

```
struct fs_struct *fs;
```

The **files** variable is pointed to opened file of a process.

```
struct files_struct *files;
```



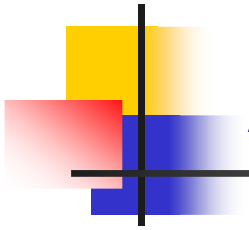
Linux Kernel : Process

Task structure *(cont)*

- Signal handlers

```
struct signal_struct *sig;
unsigned long signal, blocked;
struct signal_queue *sigqueue, **sigqueue_tail;
```

The **sig** is a signal handler. The **signal** variable contains a bit mask for signals received for the process. The **blocked** contains a bit mask for all the signals the process intends to handle later.



Linux Kernel : Process

Task structure *(cont)*

- Timing

```
unsigned long start_time;
unsigned long it_real_value, it_prof_value,
it_virt_value;
unsigned long it_real_incr, it_prof_incr, it_virt_incr;
```

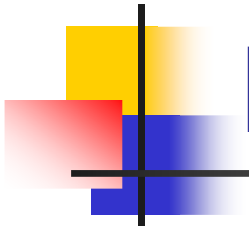
The **start_time** contains the time at which the current process was generated. The **it_real_value, it_prof_value, it_virt_value, it_real_incr, it_prof_incr** and **it_virt_incr** are used in the interval timer (interrupt).

Process Scheduling

- Linux uses two process-scheduling algorithms:
  - A time-sharing algorithm for fair preemptive scheduling between multiple processes
  - A real-time algorithm for tasks where absolute priorities are more important than fairness
- A process's scheduling class defines which algorithm to apply.
- For time-sharing processes, Linux uses a prioritized, credit based algorithm.
  - The crediting rule

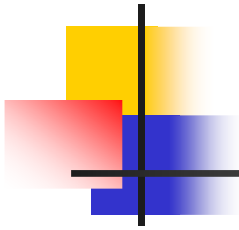
$$\text{credits} := \frac{\text{credits}}{2} + \text{priority}$$

    factors in both the process's history and its priority.
  - This crediting system automatically prioritizes interactive or I/O-bound processes.



Process Scheduling (Cont.)

- Linux implements the FIFO and round-robin real-time scheduling classes; in both cases, each process has a priority in addition to its scheduling class.
  - The scheduler runs the process with the highest priority; for equal-priority processes, it runs the process waiting the longest
  - FIFO processes continue to run until they either exit or block
  - A round-robin process will be preempted after a while and moved to the end of the scheduling queue, so that round-robing processes of equal priority automatically time-share between themselves.
- Only the root user can change the class of the current task via the **sched_setscheduler** syscall.



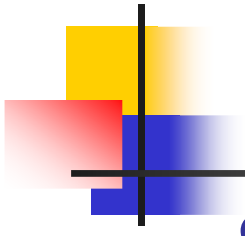
Linux Kernel : Scheduler

Scheduling classes

There are three classes of processes. (all are defined in ***include/linux/sched.h***)

```
#define SCHED_OTHER     0
#define SCHED_FIFO      1
#define SCHED_RR        2
```

- **SCHE_OTHER** : task are normal user tasks *(default)*
- **SCHE_FIFO** : task that running with this policy will never be preempted. It is a real-time policy.
- **SCHE_RR** : this is an another real-time policy, but the task will leave the CPU if there is another real-time task in the run queue.

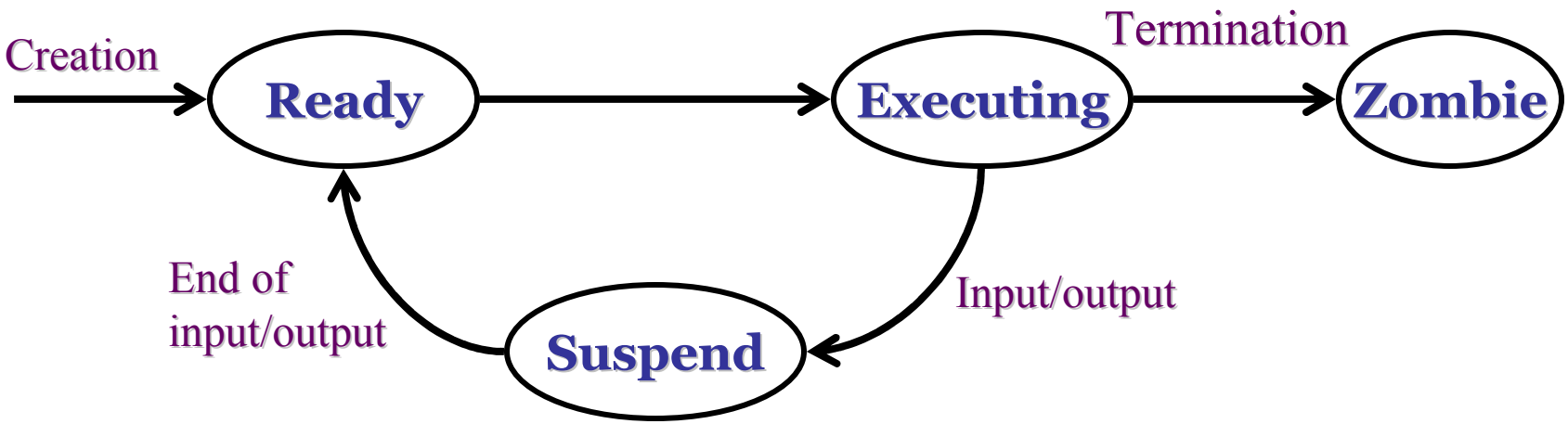


Linux Kernel : Scheduler

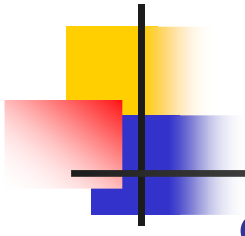
Scheduling classes *(cont)*

- **First In First Out (SCHED_FIFO)**
  - Another process of the type FIFO having a higher priority become ready, and is then executed.
  - The process becomes suspended whilst waiting for an event, such as an input or output.
  - The process voluntarily gives up the processor.

Linux Kernel : Process



Process status of SCHED_FIFO

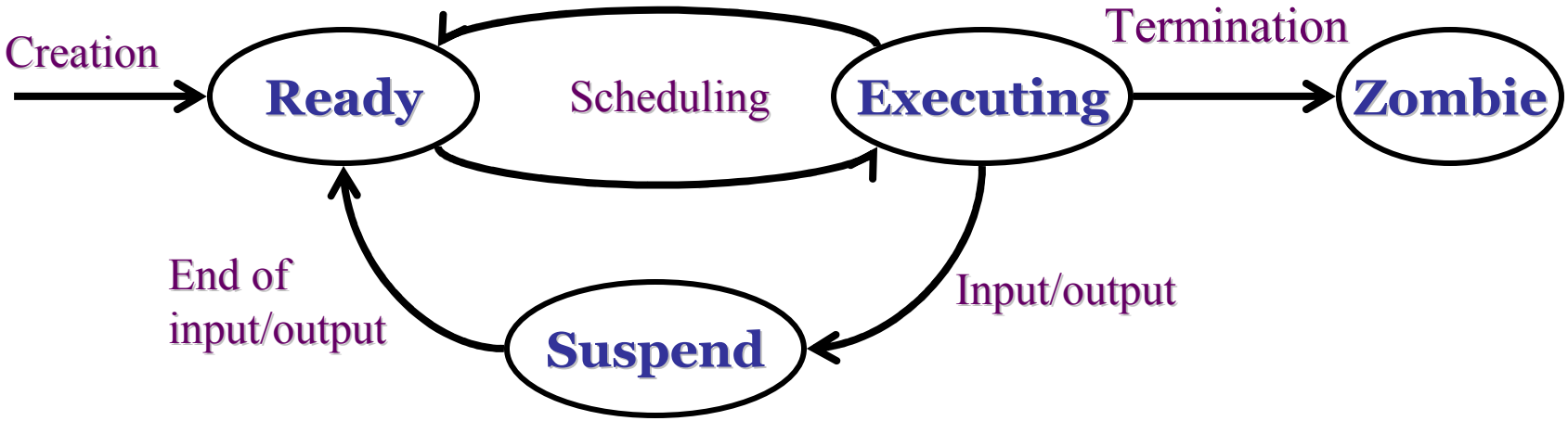


Linux Kernel : Scheduler

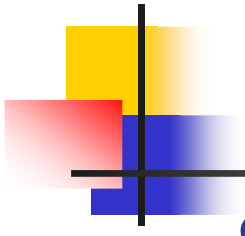
Scheduling classes *(cont)*

- **Round robin (SCHED_RR)**
  - All process have the same FIFO priority.
  - The process becomes suspended whilst waiting for an event, such as an input or output.
  - The process will leave the CPU if it takes a running time out, call ‘quantum time’.
  - The last running process is moved to the end of running queue.

Linux Kernel : Process



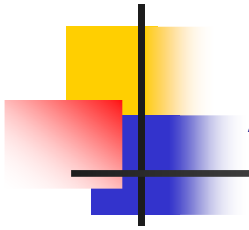
Process status of SCHED_RR



Linux Kernel : Scheduler

Scheduling classes *(cont)*

- **Other (SCHED_OTHER)**
  - The **SCHED_OTHER** is a default scheduling policy.
  - The process will be executed if no any process that is **SCHED_FIFO** or **SCHED_RR** in the running queue.
  - The process priority is used to make a decision for select the next process.



Linux Kernel : Process

Reference

- Linux Internal. Moche Bar. McGraw-Hill Companies, Inc. 2000
- Linux Kernel Internals. Michael Beck, Harald Bohme, Mirko Dziadzka, Ulrich Kunitz, Robert Magnus and Dirk Verworner. Addison-Wesley Longman Ltd. 1996
- Understanding the LINUX KERNEL.Daniel P. Bovet & Marco Cesati. O'Reilly & Associates Inc. 2001.