Operating Systems : Processes & Scheduling

Gabriel Laskar <gabriel@lse.epita.fr>

Outline

- vocabulary & generalities
- process scheduling
- process manipulation
- Inter process communication
- multithreading

LSE Security System

What is a process ?

- program: static object that contain code
- processus: program in execution
- context: address space, registers, and other infos

LSE Security System

The “OS API”

- Program respect a specific file format (ELF, MACH-O, PE)
- The kernel expose syscalls (mostly)
- Libraries expose functions

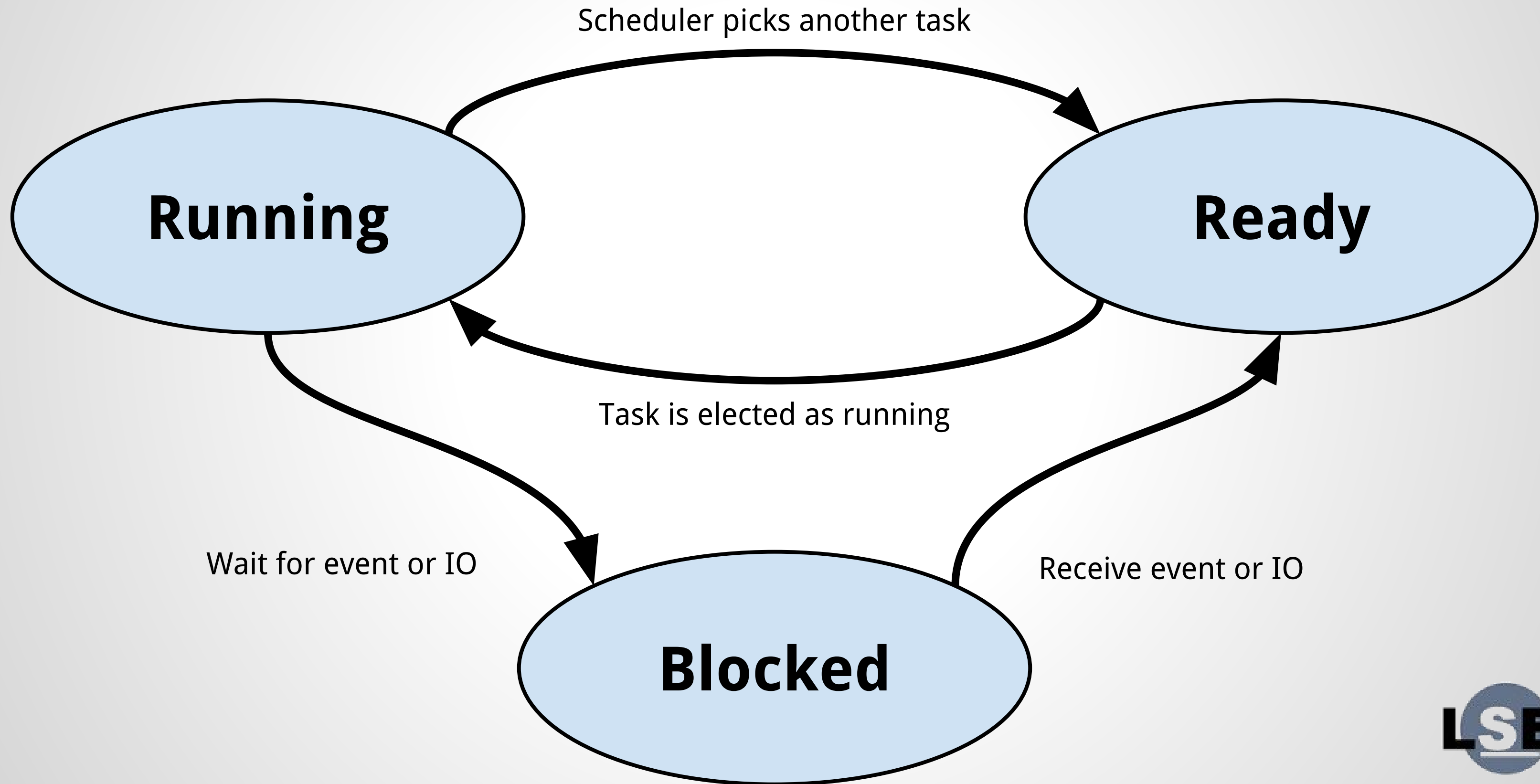
LSE Security System

Process Control Block

- struct task_struct in linux, PEB on Windows
- contains all the useful state for a task
  - state
  - stack
  - scheduling attributes
  - memory mapping
  - pid/gid/tgid
  - registers (in struct thread_info)

LSE Security System

Task states



LSE Security System

Scheduling

- process table
- queue with ready processes
- queues with blocked processes

LSE Security System

Different kind of schedulers

- long term: plan for tasks in the future
- short term: plan for next task based on dynamic informations
- middle term: based on current load, plan for actions (swapping for example)

LSE Security System

Process Creation

```
pid_t fork(void);

long clone(unsigned long flags, void *child_stack,
           void *ptid, void *ctid,
           struct pt_regs *regs);

int execve(const char *filename, char *const argv[],
           char *const envp[]);
```

LSE Security System

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

int main(int argc, char **argv, char **envp)
{
	char *prog_argv[] = { "/bin/sh", "-c", "echo is it me you looking for", NULL };
	int status;
	pid_t pid = fork();

	switch (pid) {
		case -1:
			err(1, "unable to fork");
		case 0:
			execve(prog_argv[0], prog_argv, envp);
			err(1, "unable to execve %s", prog_argv[0]);
		default:
			waitpid(pid, &status, 0);
	}
	return 0;
}
```

LSE Security System

Process hierarchy

- Unix/linux: process lives in a hierarchy
- multiple groups (signals, resource groups, ...)
- Windows: less obvious, but still some kind of tree

LSE Security System

Process Manipulation

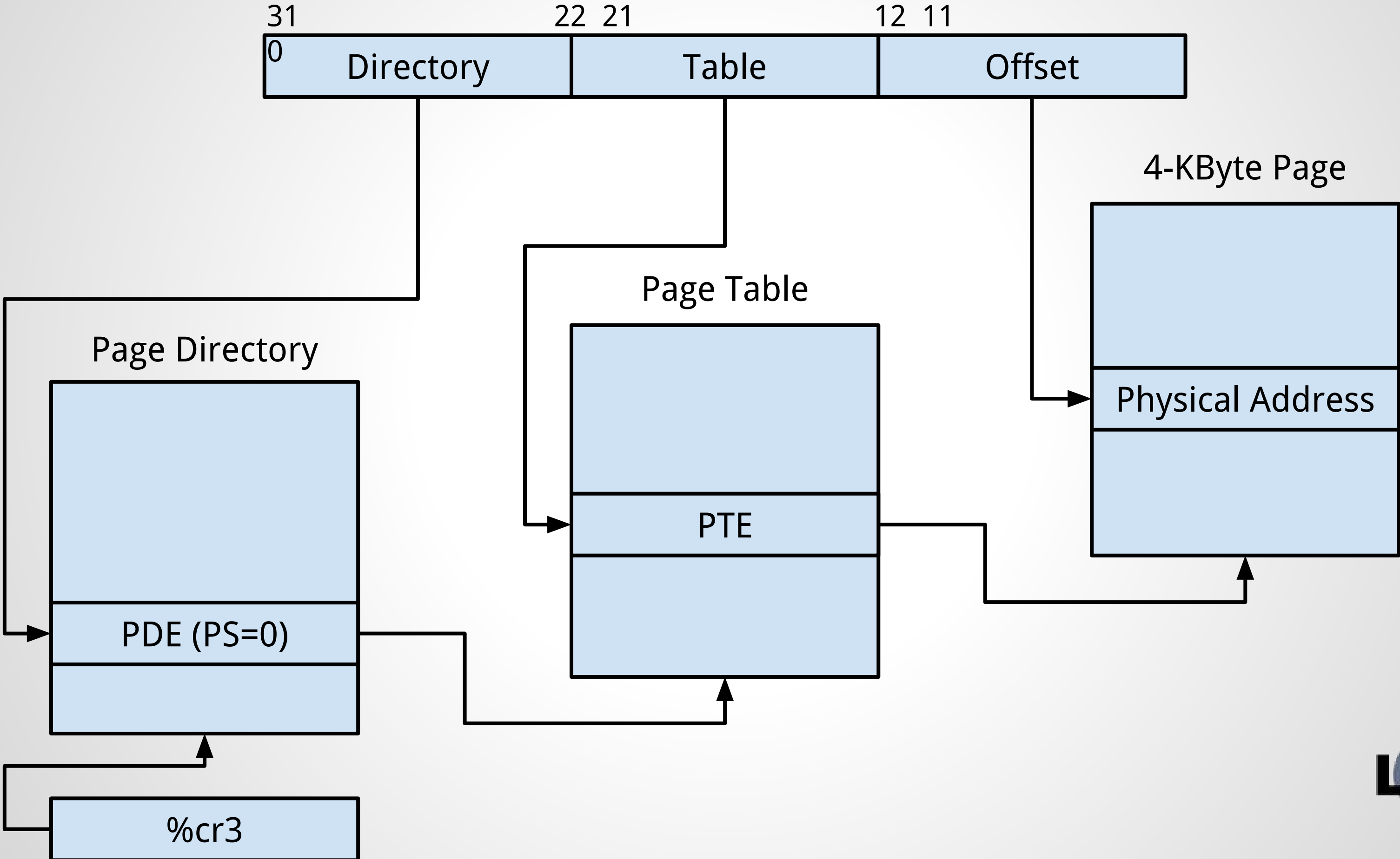
- int kill(pid_t pid, int sig);
- sighandler_t signal(int signum, sighandler_t handler);
- int sigaction(int signum, const struct sigaction *act,
  struct sigaction *oldact);

LSE Security System

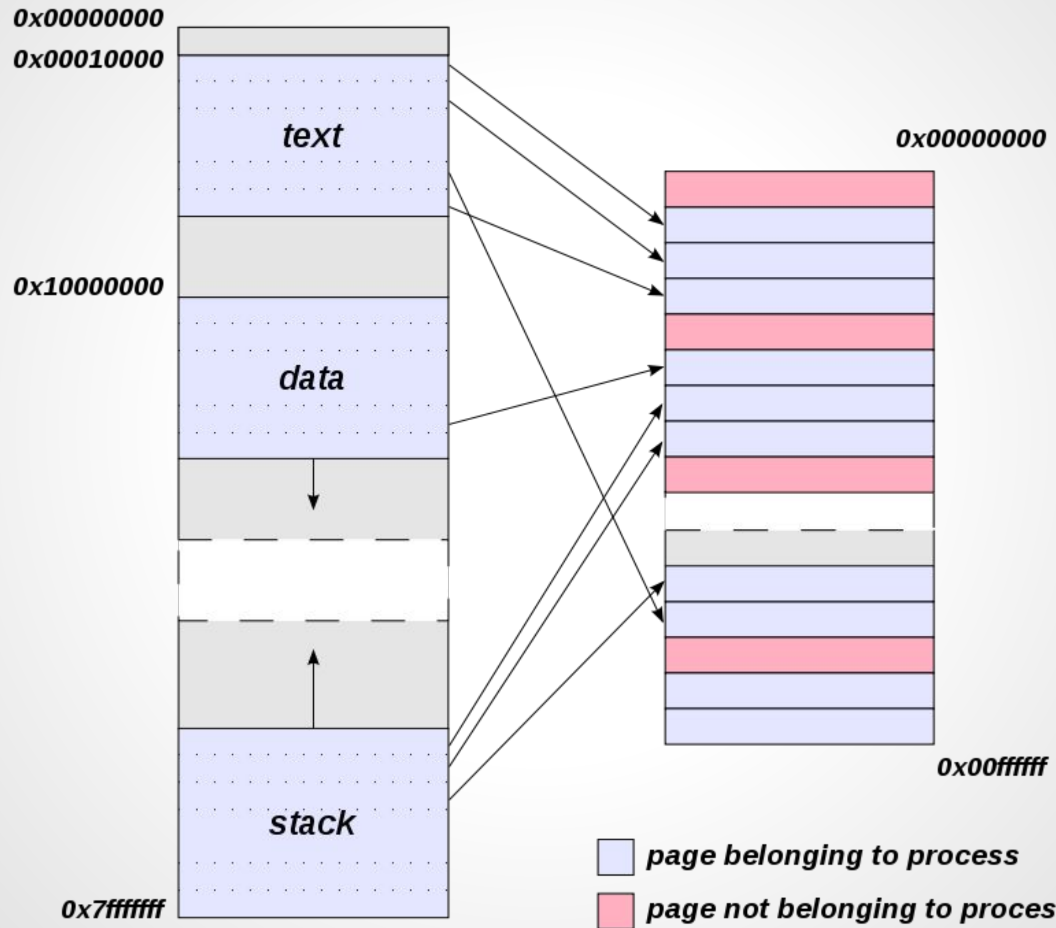
Memory Virtualization

- In the CPU
  - Memory Management Unit (MMU)
  - Page Table/Page Directory: contains memory mappings
  - Page Directory Base Pointer (PDBR): address to an address space
- In the OS
  - 1 PDBR per task => isolated address space

LSE Security System



LSE Security System


Virtual address space
Physical address space
0x00000000
0x00010000
text
0x10000000
data
stack
0x7fffffff
0x00000000
0x00ffffff
page belonging to process
page not belonging to process

LSE
Security
System

Communication between processes

- Cooperatives processes needs to communicate
  - shared memory
  - message passing
- Issues
  - how to establish a link
  - how many processes per link
  - how many link per processes

LSE Security System

Explicit communication

- automatic link created by the system
- 2 processes per liaison
- uni or bidirectional link
- Modes
  - Symmetric link
  - Asymmetric link
- Issues
  - Need to identify the process by name
  - Issue when naming changes

Indirect Communication

- link established with processes sharing the same port
- multiple processes per link
- uni or bidirectional link
- Issues
  - Multiple reception of the same message

LSE Security System

Bufferisation

- **no bufferisation:** no messages in flight, explicit synchronisation is needed
- **limited buffering:** if the buffer is full, producer is stalled
- **infinite capacity:** producer is never stalled
- **Synchronisation**
    - communication is asynchronous (no way to know if the message has been received)
    - synchronisation can be done via acknowledgment
    - possible synchronisation with blocking calls waiting for an ack

LSE Security System

Issues

- A process send messages to a terminated task
- Lost messages
- Corrupted messages

LSE Security System

Multithreading

- Problems
  - Allow parallelism inside a process
  - reduce the cost of context switching
- Solution
  - Thread (lightweight process): state, registers & stack. share other resources
  - Process: group of threads. Classical process = process with only 1 thread
- Functionalities
  - Same as a process: creation, termination, state, etc…
  - New issues: concurrent access on shared resources

LSE Security System

Userland Threads

- Principle
  - implemented as a library in userland
  - 1 thread table per process
- Pros
  - usable on a system without support for threads
  - fast context switching (no kernel trap)
  - customisable scheduling algorithm
- Cons
  - Need for unblocking syscalls
  - Threads can lock the cpu (need to yield explicitly)
  - Threads are used to alleviate blocking

LSE Security System

Kernel Threads

- Principle
  - add a thread table inside the process table
  - every blocking call is implemented as a syscall
- Pros
  - ease to create an application using them
  - no need for non blocking calls
- Cons
  - Creation/deletion/bookkeeping have a cost
  - interrupt & blocking syscalls

LSE Security System

Pthread

- POSIX api used to run threads
- Simple unified interface for multi threaded environment on POSIX system

LSE Security System

What is per-thread ?

Per Thread

- Thread ID
- signal mask
- errno
- scheduling policy
- capabilities
- CPU affinity

Per Process

- process ID
- parent Process ID
- process group
- user/group id
- file descriptors
- umask
- current directory
- limits
- ...

LSE Security System

What is scheduling ?

LSE Security System

when to schedule ?

- blocked process
- terminated (or killed) process
- new process spawn
- blocked process becomes ready

LSE Security System

Types of schedulers

- Cooperative: only blocked or terminated processes
- Preemptive: all types of events. Needs for hardware support

LSE Security System

Scheduling criterias

- different criteria to consider when trying to select the "best" scheduling algorithm
  - CPU utilization
  - Throughput
  - Turnaround time
  - Waiting time
  - Response time

LSE Security System

Types of tasks

- Interactive
  - response time: delay between submission and resolution of a request
  - wait time: time passed in ready state
- Real Time
  - Respect of deadlines
  - Predictability

LSE Security System

FCFS First come First served

- Pros:
  - No preemption
  - fifo for ready processes
  - easy to learn and understand
- Cons:
  - Great variance in scheduling criteria
  - Accumulation effect
- Bad for Shared Time System
- OK/Good for Batch Systems
- SCHED_FIFO

LSE Security System

Round Robin

- Same thing as FIFO, with a base time quantum
- Same Pros & Cons
- A little bit better for shared time systems

LSE Security System

Multiple Priority Queue

- Split tasks into multiple priorities
- Different Scheduling policy for each priority
- Scheduling between the different priorities

LSE Security System

Lottery Scheduling

- Num of ticket by task == priority
- Get a random ticket number
- Schedule the process that own the ticket
- Implementation?

LSE Security System

Completely Fair scheduling

- Try to give the same amount of power for each processes
- Count with a fair clock the “waiting time”
- Higher priority = Time elapse faster
- Store processes by “waiting time” in a Red Black Tree
- Current Linux Scheduler

LSE Security System

Real Time Scheduling

- Hard Real time need deadlines
- Soft Real time needs high priorities & small response time
- Priority inversion

LSE Security System

sched(7)

- sched_setscheduler(2)
- sched_getscheduler(2)
- sched_yield(2)
- SCHED_FIFO: First in-first out scheduling
- SCHED_RR: Round-robin scheduling
- SCHED_DEADLINE: Sporadic task model deadline scheduling
- SCHED_OTHER: Default Linux time-sharing scheduling
- SCHED_BATCH: Scheduling batch processes
- SCHED_IDLE: Scheduling very low priority jobs

LSE Security System