Major Requirements of an Operating System

- Interleave the execution of the number of processes
  - maximize resource utilization
  - provide reasonable response time
- Allocate resources to processes
- Provide facilities:
  - creation of processes by users
  - inter-process communication
  - ….

Giorgio Richelli
giorgio_richelli@it.ibm.com

The Process (abstraction)

- Also called a <task>
- Execution of an individual program
- Composed by:
  - an executable program
  - associated data
  - execution context
- It can be traced
  - list the sequence of instructions that execute

The Process (UNIX)

- Lifetime:
  - `fork()/vfork()→ exec()→ exit()`
- Well-defined hierarchy:
  - parent,child
  - orphans
    - the parent process is terminated
    - Inherited by <init>
- System processes:
  - <init> is the most important
  - Kernel threads (memory mgt, I/O, etc)

Giorgio Richelli
giorgio_richelli@it.ibm.com

MGT - [~/MasterINFN]

File New Tab Edit Settings Help

```
[giorgio@gastone MasterINFN]$ cat p.c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
    int pid, ppid, pid1;
    printf("My ID = %d\n",getpid());
    printf("My Parent ID = %d\n",getppid());
    return 0;
}
[giorgio@gastone MasterINFN]$ cc p.c
[giorgio@gastone MasterINFN]$ ps
  PID TTY          TIME CMD
 8582 pts/2    00:00:00 bash
18230 pts/2    00:00:00 ps
[giorgio@gastone MasterINFN]$ a.out
My ID = 18234
My Parent ID = 8582
[giorgio@gastone MasterINFN]$
```

~/MasterINFN | Root | ~/MasterINFN

Giorgio Richelli
giorgio_richelli@it.ibm.com

Scheduler

- A kernel service that assign the processor to a process, based on policies & priorities
- It should prevents a single process from monopolizing processor time
- It cannot just select the process that has been in the queue the longest, e.g. it may be blocked (waiting for an event such as I/O, etc.)

Giorgio Richelli
giorgio_richelli@it.ibm.com

Process Creation

- A process is always spawned by another, existing, process (the <parent>)

- E.g.:
  - Job Scheduler (submission of a batch job)
  - Upon user logon (getty)
  - By OS, to provide a system service, such as printing

Giorgio Richelli
giorgio_richelli@it.ibm.com

Process Termination

Reason:

- Voluntarily:
  - the process executes a request to terminate (`exit()`)
- Killed by the system:
  - On error and fault conditions

Giorgio Richelli
giorgio_richelli@it.ibm.com

Reasons for (abnormal) Termination

- Time limit exceeded
- Memory unavailable
- Bounds violation
- Arithmetic error
- Timer overrun (e.g. SIGALRM)
  - process waited longer than a specified maximum for an event

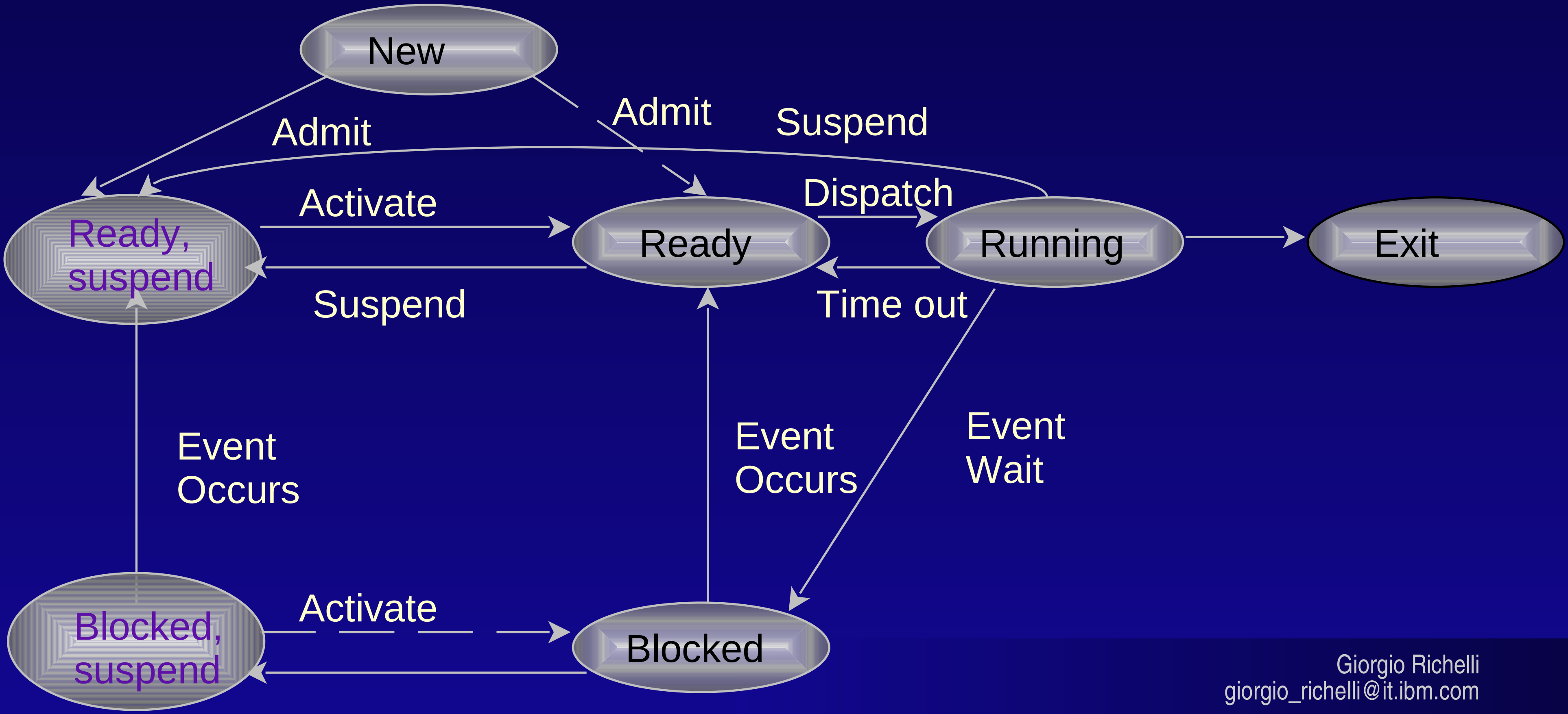
Giorgio Richelli
giorgio_richelli@it.ibm.com

Reasons for (abnormal) Termination

- Invalid instruction
  - e.g trying to execute data
- Privileged instruction
- Operating system intervention (such as when deadlock occurs)
- Parent terminates → child processes may be terminated
- Request by another process (`kill()`)

Giorgio Richelli
giorgio_richelli@it.ibm.com

Process State Transition Diagram with Two Suspend States



Giorgio Richelli
giorgio_richelli@it.ibm.com

Process Creation

- Assign a unique process identifier
- Allocate space for the process
- Initialize process control blocks
- Set up appropriate linkages, e.g:
    - add new process to linked list used for scheduling queue
    - maintain an accounting file
    - ..

When to Switch a Process

- Interrupts
  - Clock (time slice expired)
  - I/O
- Memory fault
  - memory page is not mapped
- Trap (sw interrupt)
  - error occurred
  - may cause process to be moved to <Exit> state
- System call (also a sw int.)
  - such as open()

Giorgio Richelli
giorgio_richelli@it.ibm.com

UNIX Process State

- Initial (idle)
- Ready to run
- Kernel/User running
- Zombie
- Asleep

+ (4BSD): stopped/suspend

Giorgio Richelli
giorgio_richelli@it.ibm.com

Process states and state transitions

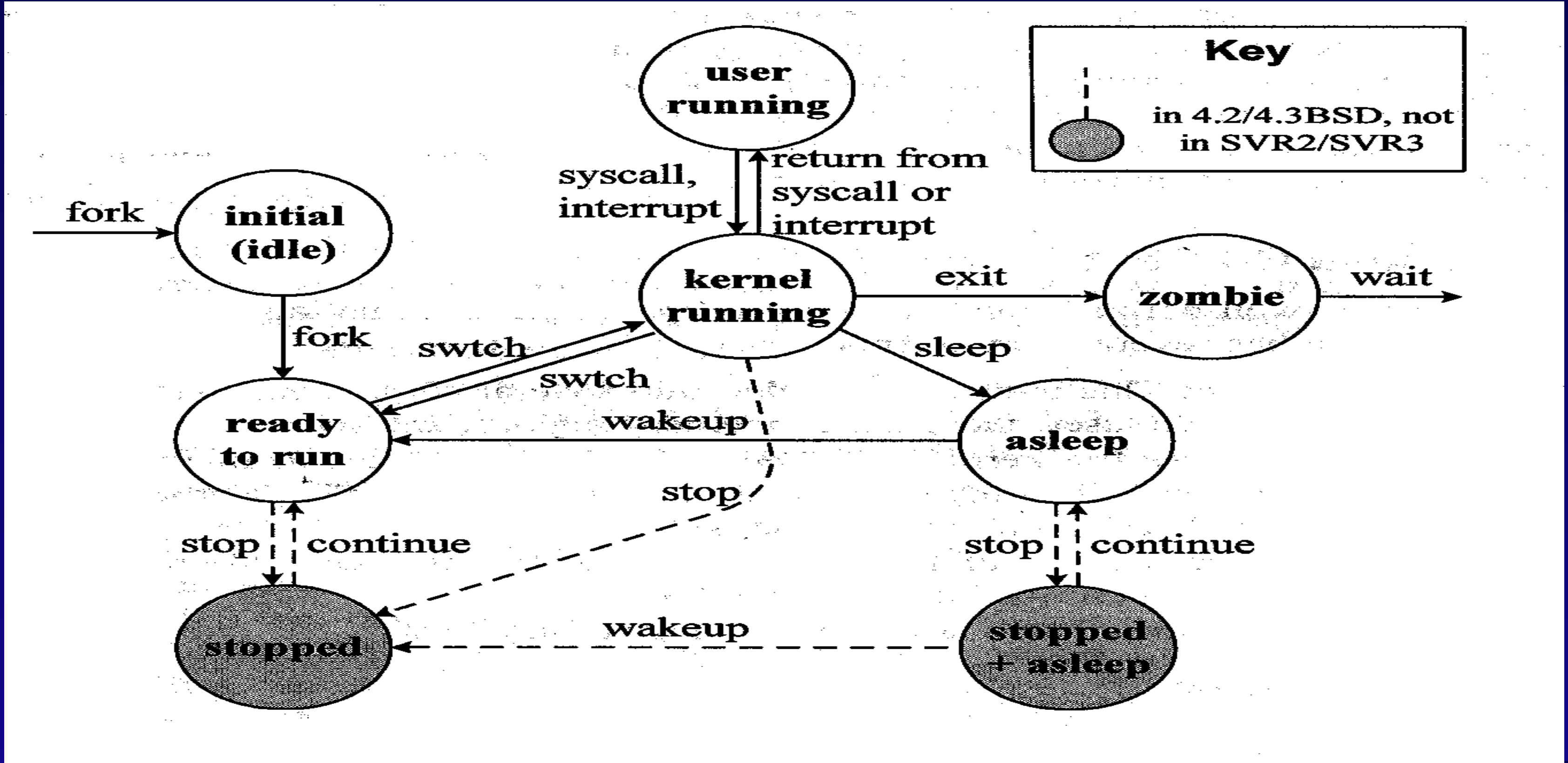


Figure 2-3. Process states and state transitions.

Process Context

- User address space:
  - code, data, stack, shared memory regions
- Control information:
  - u area, proc, kernel stack, addr.trans. map
- Credentials: UID & GID
- Environment variables:
  - inherited from the parent
- Hardware context(in PCB of u area):
  - PC, SP, PSW, MMR, FPU

User Credentials

- UserID, GroupID
  - Superuser: UID=0, GID=0
- Real IDs:
  - login, send signals
- Effective IDs:
  - file creation and access
- exec():
  - suid/sgid mode: set to that of the <owner of the file>

Giorgio Richelli
giorgio_richelli@it.ibm.com

Who's who

- `int getuid();`
  - returns userid
- `int getgid()`
  - returns groupid
- `int geteuid();`
  - return <effective> userid
- `int getegid();`
  - returns <effective> groupid

Giorgio Richelli
giorgio_richelli@it.ibm.com

A typical process hierarchy in 4.3BSD UNIX

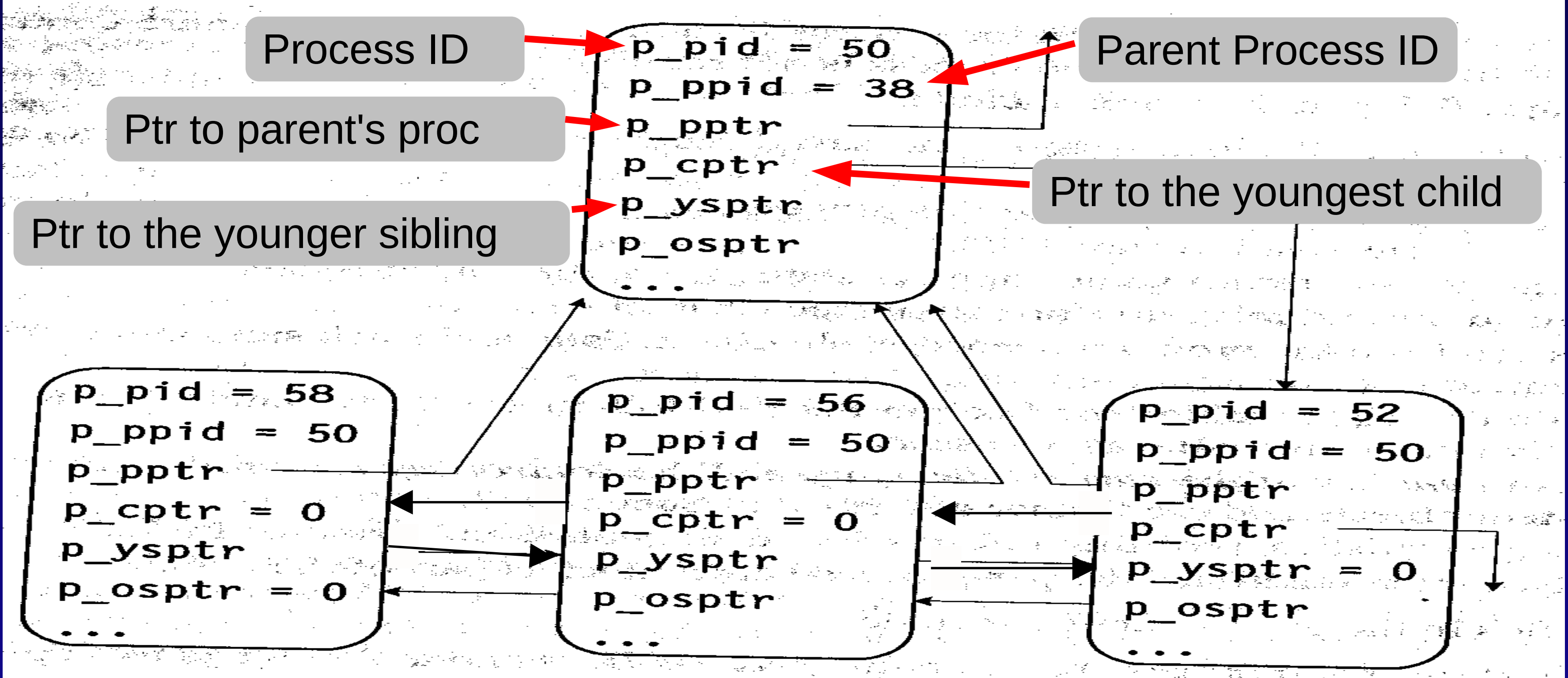


Figure 2-4. A typical process hierarchy in 4.3BSD UNIX.

giorgio_richelli@it.ibm.com

The UNIX kernel

- A special program that runs directly on the hardware.
- Implements the process model and services.
- Resides on disk
- /vmunix, /unix, /vmlinuz, ...
- Bootstrapping: loads the kernel.
- Initializes the system and sets up the environment, remains in memory before shut down

Giorgio Richelli
giorgio_richelli@it.ibm.com

UNIX Kernel Services

- Provides System Calls
- Interfaces with hardware devices
- Manages exceptions
  - Divide by 0, overflowing user stack
- Handles Interrupts
- Implement other facilities (vm management, networking, ..)

Giorgio Richelli
giorgio_richelli@it.ibm.com

The Kernel interacts with processes and devices

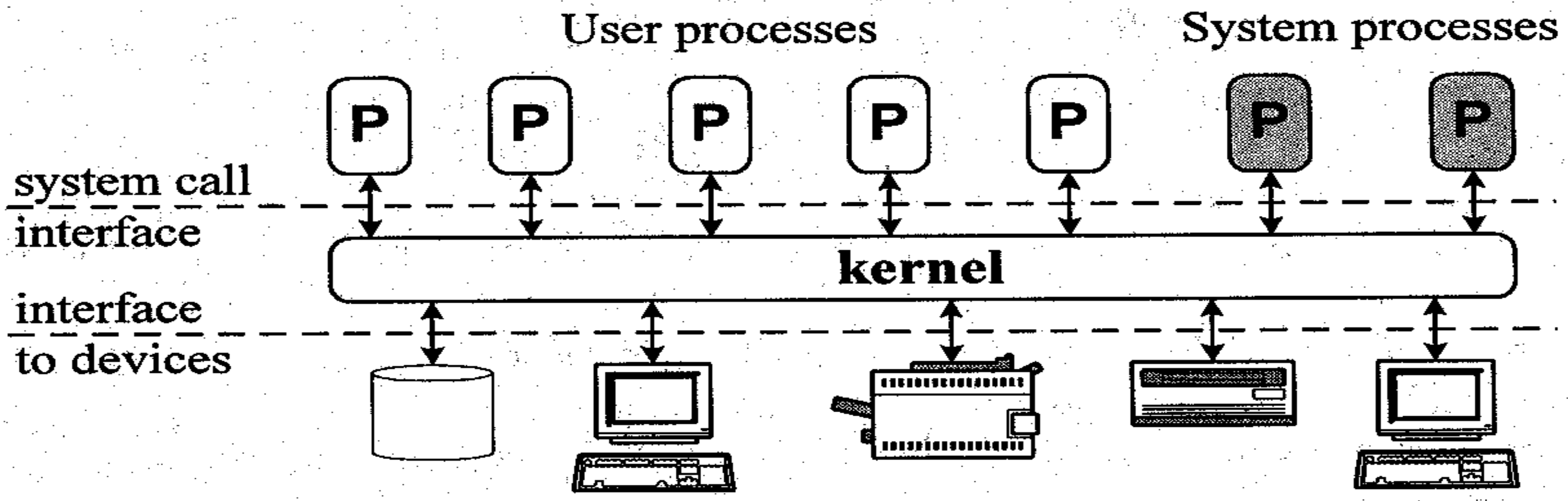


Figure 2-1. The kernel interacts with processes and devices.

Giorgio Richelli
giorgio_richelli@it.ibm.com

Mode,Space & Context

- Some critical resources must be protected
- Virtual Memory
  - VM space
  - Address Translation Maps
  - Memory Management Unit

- Kernel Mode
  - more privileged, kernel functions
- User Mode
  - less privileged, user functions

Giorgio Richelli
giorgio_richelli@it.ibm.com

Kernel data

- One instance of the kernel
  - kernel stack
- Per-process objects
  - info. about a process

- Global data structures
- Current process

- System call → mode switch

Giorgio Richelli
giorgio_richelli@it.ibm.com

Context

- Must be re-entrant
  - several processes may be involved in kernel activities concurrently.
- Execution context
  - Process
  - System (Interrupt)

Giorgio Richelli
giorgio_richelli@it.ibm.com

Executing in Kernel Mode

- Different types of events:
  - Device interrupts
  - Exceptions
  - Traps or software interrupts
- Dispatch table

- System context: interrupts
- Process context: traps, exceptions & software interrupts

Giorgio Richelli
giorgio_richelli@it.ibm.com

Execution mode and Context

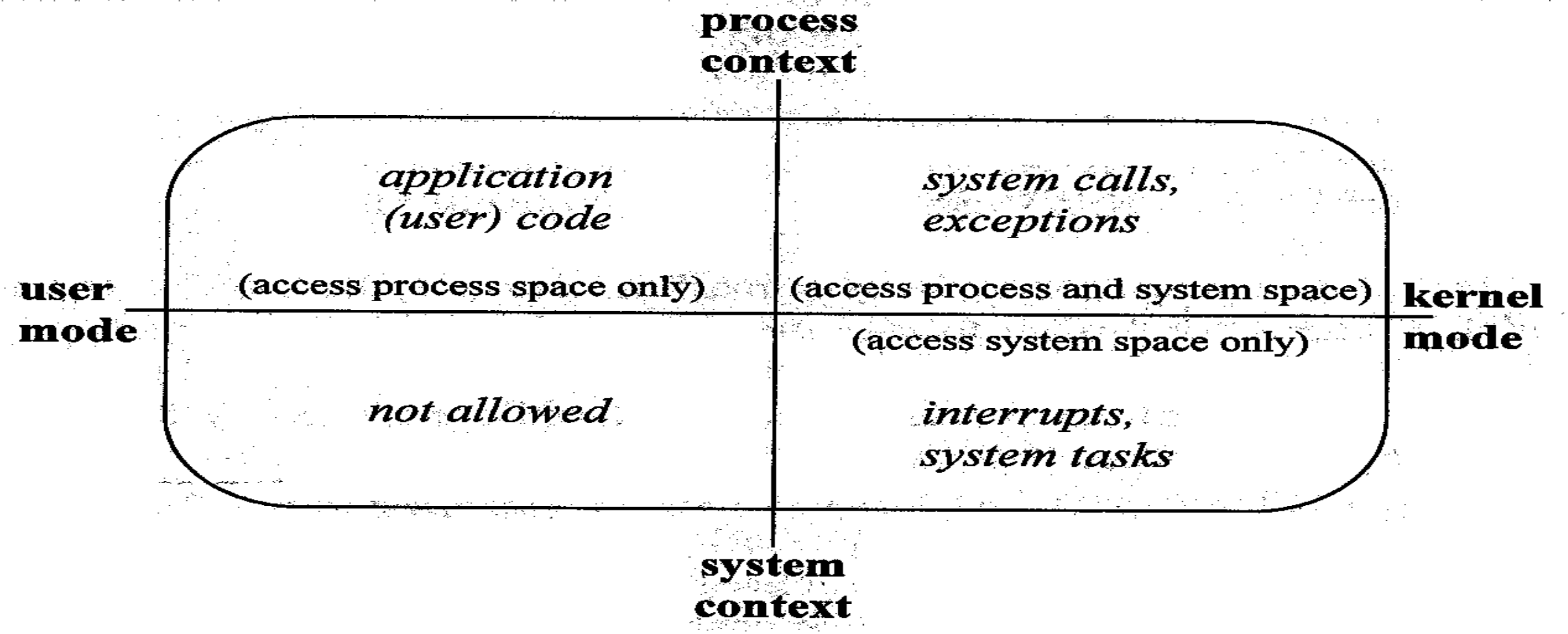


Figure 2-2. Execution mode and context.

Giorgio Richelli
giorgio_richelli@it.ibm.com

The System Call Interface

- `syscall()`
  - kernel mode
  - process context
  - Copy arguments , save hardware context on the kernel stack
  - Use system call number to index dispatch vector
  - Return results in registers, restore hardware context, user mode, control back to the library routine.

Giorgio Richelli
giorgio_richelli@it.ibm.com

New Processes & Programs

`int fork()`

- creates a new process.
- returns 0 to the child, PID to the parent

`int exec*(..)`

- begins to execute a new program

Giorgio Richelli
giorgio_richelli@it.ibm.com

Using fork & exec

```
if ((ChildPid = fork())==0){
   /* child code*/
   … …
    if (execve(“new program”),…)<0) {
        perror(“execve failed.”);
        exit(-1)
    }
} else if (ChildPid <0) {
        perror(“fork failed”);
        exit(-1)
}
/*parent continues here*/
```

Giorgio Richelli
giorgio_richelli@it.ibm.com

MGT - [~/MasterINFN]

File New Tab Edit Settings Help

```
[giorgio@gastone MasterINFN]$ cat p2.c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
    int pid, ppid, pid1;

    pid=fork();
    if (pid==0) {
        printf("Child Process - My PID:%d, Parent PID:%d\n", getpid(), getppid());
    }
    else {
        printf("Parent Process- My PID:%d, Parent PID:%d\n", getpid(), getppid());
    }
    return 0;
}
[giorgio@gastone MasterINFN]$ ps
  PID TTY          TIME CMD
 8582 pts/2    00:00:00 bash
18351 pts/2    00:00:00 ps
[giorgio@gastone MasterINFN]$ cc p2.c
[giorgio@gastone MasterINFN]$ a.out
Parent Process- My PID:18364, Parent PID:8582
Child Process - My PID:18365, Parent PID:18364
[giorgio@gastone MasterINFN]$
```

~/MasterINFN Root ~/MasterINFN

Giorgio Richelli
giorgio_richelli@it.ibm.com

Process Creation

Creates (almost) an exact clone of the parent:

- ✔ Reserve swap space for the child
- ✔ Allocate a new PID and proc structure for the child
- ✔ Initialize proc structure
- ✔ Allocate ATM (address translation map)
- ✔ Allocate u_area and copy from parent
- ✔ Update the u_area to refer to the new ATM & swap space
- ✔ Add the child to the set of processes sharing the text region of the program

Giorgio Richelli
giorgio_richelli@it.ibm.com

Process Creation

- Duplicate the parent's data and stack regions update ATM to refer to these new pages.
- Acquire references to shared resources inherited by the child
- Initialize the hardware context
- Make the child runnable and put it on a scheduler queue
- Arrange to return 0 to child
- Return the PID to the parent

Fork Optimization

- It is wasteful to immediately make a copy of the address space of the parent
- Copy-on-write:
  - only the pages that are modified must be copied
- `vfork()`:
  - parent loans the address space and blocks until the child returns

Giorgio Richelli
giorgio_richelli@it.ibm.com

Invoking a New Program

Process address space:

- ✓ Text (code)
- ✓ Initialized data
- ✓ Uninitialized data
- ✓ Shared memory
- ✓ Shared libraries
- ✓ Heap
- ✓ Stack

Giorgio Richelli
giorgio_richelli@it.ibm.com

Awaiting Process Termination

```
pid_t wait(int *wstatus);

pid_t waitpid(pid_t pid, int *wstatus, int options);

int waitid(idtype_t idtype, id_t id, siginfo_t *infop,
int options);
```

Giorgio Richelli
giorgio_richelli@it.ibm.com

Zombie Processes

- Upon termination, kernel holds proc structure
- `wait()` frees the proc
  - called by parent or the init process.

- When:
  - child dies before the parent
  - parent doesn't wait for all childs,

  the proc is never released.

Giorgio Richelli
giorgio_richelli@it.ibm.com

Zombie Processes

Scenario:

- child exits -> [Defunct]
- parent doesn't wait() for child & ignores SIGCHLD → zombie
- eventually parent exits → child is inherited by init → proc freed

Giorgio Richelli
giorgio_richelli@it.ibm.com