

# Schema chiamate sistema

## Libreria NPTL

pthread\_create  
mutex\_lock  
sem\_wait  
pthread\_join  
mutex\_unlock  
sem\_post  
pthread\_exit  
return

## Clib

fopen  
fread  
fwrite

## glibc

clone  
fork  
exec\*  
exit  
wait/waitpid  
nanosleep  
open  
read  
write  
void syscall(syscall\_number, ...) {  
SYSCALL Passaggio modo U->S  
}

void system\_call() {}

## System call service routines

fork, clone	→ sys_clone	exec*	→ sys_execve
mutex_lock, sem_wait, pthread_join	→ sys_futex(wait)	wait/waitpid	→ sys_wait
mutex_unlock, sem_post	→ sys_futex(wakeup)	nanosleep	→ sys_nanosleep
return, pthread_exit	→ sys_exit	open	→ sys_open
exit	→ sys_exit_group	read	→ sys_read
		write	→ sys_write

# Moduli del sistema operativo

## System call

```
system_call(...) {
    call a servizio opportuno();
    if (modo_rientro == U &&
    ↪ TIF_NEED_RESCHED == 1) {
        schedule();
    }
    SYSRET
}
```

## Interrupt routines

```
void R_int_clock(...) {
    // con periodicità opportuna
    CURR->sched_class->task_tick();
    // con periodicità opportuna
    Controlla_timer();
    if (modo_rientro == U &&
    ↪ TIF_NEED_RESCHED == 1) {
        schedule();
    }
    IRET
}
```

```
void R_int_X(...) {
    wakeup(...);
    if (modo_rientro == U &&
    ↪ TIF_NEED_RESCHED == 1) {
        schedule();
    }
    IRET
}
```

## Service routines

```
sys_clone(...) {
    check_preempt_curr();
}
```

```
sys_nanosleep(...) {
    curr->state = ATTESA;
    schedule_timeout();
}
```

```
sys_futex(wait, ...) {
    wait_X(...);
}
```

```
sys_futex(wakeup, ...) {
    wake_up_process(padre);
}
```

```
sys_execve(...) {
}
```

```
sys_read/write/open(...) {
    wait_X(...);
}
```

```
sys_wait(...) {
    curr->state = ATTESA;
    schedule();
}
```

```
sys_exit(...) {
    wake_up_process(padre);
    schedule();
}
```

```
sys_exit_group(...) {
    for (tutti i processi
    ↪ con stesso TGID) {
        sys_exit(...)
    }
}
```

## Gestione dello stato

```
wake_up/wake_up_process(...) {
    for (ogni task in waitqueue) {
        task->state = PRONTO;
        enqueue_task(task); //NN
        if (flag==EXCLUSIVE) break;
    }
    check_preempt_curr();
}
```

```
wait_X(...) {
    CURR->state = ATTESA;
    schedule();
}
```

## Gestione tempo

```
schedule_timeout(...) {
    add_timer(); //NN
    schedule();
    del_timer(); //NN
}
```

```
Controlla_timer(...) {
    for (ogni timeout creato) {
        if (timeout scaduto) {
            wakeup_process(...);
        }
    }
}
```

## Routine dello scheduler

```
schedule(...) {
    prev = CURR;
    if (prev->stato == ATTESA ||
    prev terminato) {
        dequeue_task(prev); //NN
    }
    next = pick_next_task();
    if (next != prev) {
        // MACRO
        context_switch(prev, next)
    }
    TIF_NEED_RESCHED = 0
}
```

```
task_tick_X(...) {
    if (QdT scaduto) {
        resched();
    }
}
```

```
check_preempt_curr(...) {
    if (occorre invocare scheduler) {
        resched();
    }
}
```

```
pick_next_task(...) {
    for (ogni classe di scheduling cl,
    ↪ in ordine gerarchico) {
        task = cl->pick_next_task(); //NN
        if (task) return task;
    }
}
```

```
resched(...) {
    TIF_NEED_RESCHED = 1;
}
```