

COMP3301

Lecture 2(b)

Processes: implementation

School of Information Technology and Electrical Engineering
The University of Queensland

This week

- First half of lecture:
 - Theory about processes
- Now:
 - Down and dirty with the details
 - How things work in a real system
 - Introduce Assignment 1

Process Tree

- Not quite like the classical version!
- CLOCK and SYSTEM don't have PID
- pm is PID0
- rs is parent of all boot image processes
- init gets PID1
 - executes /etc/rc
 - starts drivers and servers
 - starts terminals

Reincarnation server

- Adopts system processes
- Can restart if necessary
- *service*

ps -xl

F S UID	PID	PPID	PGRP	SZ	RECU	TTY	TIME	CMD
0 R 0	(-4)	0	0	60		?	0:33	IDLE
10 W 0	(-3)	0	0	60	ANY	?	0:00	CLOCK
0 R 0	(-2)	0	0	60		?	0:02	SYSTEM
2 W 0	(-1)	0	0	60		?	0:00	KERNEL
10 W 0	0	5	98	92	ANY	?	0:00	pm
10 W 0	4	5	0	4928	memory	?	0:00	fs
10 W 0	5	1	0	160	ANY	?	0:00	rs
10 W 0	8	5	0	16	SYSTEM	?	0:00	memory
10 W 0	9	5	0	76	ANY	?	0:00	log
10 W 0	7	5	0	80	ANY	?	0:00	tty
10 W 0	10	5	0	56	ANY	?	0:00	driver
10 W 0	6	5	0	136	ANY	?	0:00	ds
10 S 0	1	5	0	16	(wait) pm	?	0:00	init
0 R 0	98	1	98	180		co	0:00	-sh

Minix IPC

- `send(dest, &message)`
- `receive(source, &message)`
- `sendrec(src_dst, &message)`
- implemented by kernel
- User processes can't send to each other
- User processes to servers, servers to drivers
- No buffers – sender blocks
- `notify(dest)`

Minix IPC

- User processes don't use send and receive directly
- When a system call is made:
 - sendrec happens internally
 - software interrupt (trap) happens
 - kernel handles
 - see [kernel/mpx386.s](#) and [kernel/proc.c](#)

Scheduling

- The simplest scheme is called round-robin
 - Each process is allocated a *quantum*
 - Once used, process goes to back of queue
- More complex algorithm: prioritized round-robin
 - Multiple queues
 - Higher priority queues checked first for runnable processes

Scheduling in Minix

- Multilevel queuing system
- 16 queues (by default)
 - lowest priority – IDLE
 - servers in higher priority queues than user processes
 - drivers higher still
 - don't usually use all 16 at once!
- Different quantumums for user processes (smaller than for drivers)

From /kernel/proc.h

```
/* Scheduling priorities for p_priority. Values must start at
zero (highest priority) and increment. Priorities of the
processes in the boot image can be set in table.c. IDLE must
have a queue for itself, to prevent low priority user
processes to run round-robin with IDLE.
*/
#define NR_SCHED_QUEUES 16 /* MUST equal
minimum priority + 1 */
#define TASK_Q 0 /* highest, used for kernel tasks */
#define MAX_USER_Q 0 /* highest priority for user processes */
#define USER_Q 7 /* default (should correspond to nice 0) */
#define MIN_USER_Q 14 /* minimum priority for user
processes */
#define IDLE_Q 15 /* lowest, only IDLE process goes
here */
```

Priorities

- Process can be changed to different queue by the system
- Or, user can invoke *nice*
 - see [kernel/system/do_nice.c](#)
- If user process uses quantum, priority is lowered

Scheduling algorithm

- Round-robin
 - sort of
- Processes that don't use quantum are assumed to be waiting on IO
 - they're put at the head of the queue (when ready)
- Processes that use quantum go to end of queue (round robin)
- All within the context of priority queues

Process Data structures

- Header files – textbook has details
- Process table
 - Partially replicated in fs and pm
 - Shift-F1 shows pm process table
- Each entry defined as struct proc

from kernel/proc.h

```
struct proc {
    struct stackframe_s p_reg; /* process' registers saved in stack
                               frame */
    reg_t p_ldt_sel; /* selector in gdt with ldt base and limit */
    struct segdesc_s p_ldt[2+NR_REMOTE_SEGS]; /* CS, DS and
                                               remote segments */

    proc_nr_t p_nr; /* number of this process
                    (for fast access) */

    struct priv *p_priv; /* system privileges structure */
    char p_rts_flags; /* SENDING, RECEIVING, etc. */

    char p_priority; /* current scheduling priority */
    char p_max_priority; /* maximum scheduling priority */
    char p_ticks_left; /* number of scheduling ticks left */
    char p_quantum_size; /* quantum size in ticks */
};
```

from kernel/proc.h (contd)

```
struct mem_map p_memmap[NR_LOCAL_SEGS]; /* memory map (T,
D, S) */

clock_t p_user_time; /* user time in ticks */
clock_t p_sys_time; /* sys time in ticks */

struct proc *p_nextready; /* pointer to next ready process */
struct proc *p_caller_q; /* head of list of procs wishing to send */
struct proc *p_q_link; /* link to next proc wishing to send */
message *p_messbuf; /* pointer to passed message buffer */
proc_nr_t p_getfrom; /* from whom does process want to
receive? */
proc_nr_t p_sendto; /* to whom does process want to send? */

sigset_t p_pending; /* bit map for pending kernel signals */

char p_name[P_NAME_LEN]; /* name of the process, including \0 */
};
```

Privilege table

- Table describing which processes can communicate with
- Hit F4 to see some details
- Minix uses Intel privilege levels
 - Kernel and interrupt handlers
 - Kernel tasks
 - Servers and user processes

Scheduling

- Where does it happen?
 - Have a look at [kernel/proc.c](#)

Assignment 1

- Some details...

Summary

- Scheduling
- System task
- Clock task