PandOSsh: Phase 1

Luca Bassi (luca.bassi14@studio.unibo.it)
Luca Orlandello (luca.orlandello@studio.unibo.it)

November 17, 2025

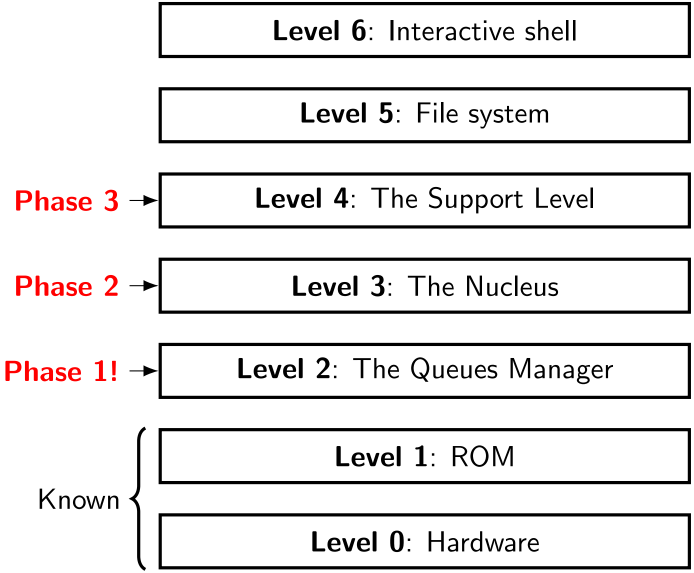
PandOSsh

- PandOSsh is an educational *multiprocessor* operating system
- Evolution of PandOSplus; they are also the evolution of a long list of O.S. proposed for educational purposes (PandOS+, Kaya, HOCA, TINA, ICARO, etc).
- PandOSsh must be realized on the μRISCV architecture
- Architecture based on six levels of abstraction, on the model of the O.S. THE proposed by Dijkstra in one of his articles of 1968

μRISCV

- Emulates a RISC-V architecture (project started in 2010 and it's evolving)
- RISC-V architecture is fully open source

PandOSsh: six levels of abstraction



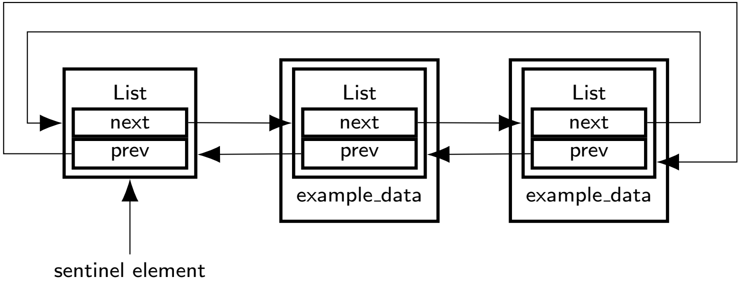
PandOSsh Phase 1 requests

- For this first phase, you have to add the functions requested, and the test procedure must conclude correctly.
- Level 2 requirements are purely logical; it is not yet touched the specific hardware of the emulator.
- You are provided with a **.tar.gz** file, that contains useful files to begin with.

The `listx` module

- A provided module, with a subset of functions of the Linux's `list.h` module
- It implements generic and type-oblivious lists
- Take a closer look to its content!!

```
struct list_head {
    struct list_head *next;
    struct list_head *prev;
}
```



Level 2 specs: PCB

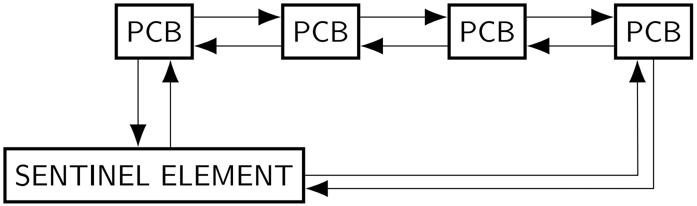
- PandOSsh level 2 (The Queues Manager) provides the implementation of the data structures used from the level 3 (The Nucleus)
- The Process Control Block (PCB) represent a process in the system.
- The queue manager implements PCB-related features:
  - Allocation and Deallocation of PCBs.
  - PCB Queue Management.
  - PCB Tree Management.
- ASSUMPTION: no more than `MAXPROC` (20) concurrent processes in PandOSsh

PCB data structure

```c
/* process table entry type */
typedef struct pcb_t {
    /* process queue  */
    struct list_head p_list;

    /* process tree fields */
    struct pcb_t    *p_parent; /* ptr to parent */
    struct list_head p_child;  /* children list */
    struct list_head p_sib;    /* sibling list  */

    /* process status information */
    state_t p_s;    /* processor state */
    cpu_t   p_time; /* cpu time used by proc */

    /* Pointer to the semaphore the process is currently blocked on */
    int *p_semAdd;
    /* Pointer to the support struct */
    support_t *p_supportStruct;
    /* Indicator of priority */
    int p_prio;
    /* process id */
    int p_pid;
} pcb_t, *pcb_PTR;
```

PCB's list

- PCBs can be organized into queues, called process queues (e.g. active process queue).
- Each list is managed via list_head fields.
- A list is identified by a sentinel element of type list_head.



PCB allocation

Main data structures:

- `pcbFree_h`: free or not used PCBs' list.
- `pcbTable[MAXPROC]`: array of PCBs with max size of `MAXPROC`.

PCB's functions to implement

Allocation and Deallocation of PCBs:

1. `void freePcb(pcb_t *p)`: insert the element pointed to by `p` onto the `pcbFree` list.
2. `pcb_t *allocPcb()`: return NULL if the `pcbFree` list is empty. Otherwise, remove an element from the `pcbFree` list, provide initial values for ALL of the PCBs fields and then return a pointer to the removed element. PCBs get reused, so it is important that no previous value persist in a PCB when it gets reallocated.
3. `void initPcbs()`: initialize the `pcbFree` list to contain all the elements of the static array of `MAXPROC` PCBs. This method will be called only once during data structure initialization.

PCB's functions to implement

PCB Queue Management:

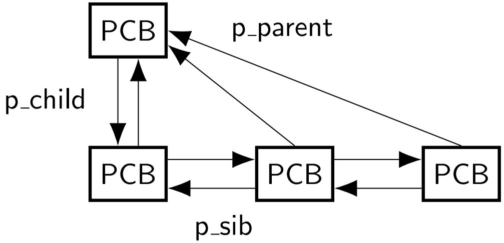
4. `void mkEmptyProcQ(struct list_head *head)`: this method is used to initialize a variable to be head pointer to a process queue.
5. `int emptyProcQ(struct list_head *head)`: return TRUE if the queue whose head is pointed to by `head` is empty. Return FALSE otherwise.
6. `void insertProcQ(struct list_head *head, pcb_t *p)`: insert the PCB pointed by `p` into the process queue whose head pointer is pointed to by `head`. The list must be ordered by priority. In case of equal priority, the new PCB must be inserted after the last PCB with this priority. For example, if you insert PCB `N` with priority `10` to the queue, `A(20)`, `B(10)`, `C(0)`, the list becomes `A(20)`, `B(10)`, `N(10)`, `C(0)`.

PCB's functions to implement

7. `pcb_t *headProcQ(struct list_head *head)`: return a pointer to the first PCB (i.e. the PCB with max priority) from the process queue whose head is pointed to by `head`. Do not remove this PCB from the process queue. Return NULL if the process queue is empty.
8. `pcb_t *removeProcQ(struct list_head *head)`: a pointer to the first PCB (i.e. the PCB with max priority) from the process queue whose head is pointed to by `head`. Return NULL if the process queue was initially empty; otherwise return the pointer to the removed element.
9. `pcb_t *outProcQ(struct list_head *head, pcb_t *p)`: remove the PCB pointed to by `p` from the process queue whose head pointer is pointed to by `head`. If the desired entry is not in the indicated queue (an error condition), return NULL; otherwise, return `p`. Note that `p` can point to any element of the process queue.

PCB's trees

- In addition to the possibility of participating in a process queue, PCBs can be organized into process trees.
- Each parent contains a pointer to the list of the children (`p_child`)
- Each child has a pointer to the parent (`p_parent`) and a pointer that connects the brothers to each other.



PCB's functions to implement

PCB Tree Management:

10. `int emptyChild(pcb_t *p)`: return TRUE if the PCB pointed to by `p` has no children. Return FALSE otherwise.
11. `void insertChild(pcb_t *prnt, pcb_t *p)`: make the PCB pointed to by `p` a child of the PCB pointed to by `prnt`.
12. `pcb_t *removeChild(pcb_t *p)`: make the first child of the PCB pointed to by `p` no longer a child of `p`. Return NULL if initially there were no children of `p`. Otherwise, return a pointer to this removed first child PCB.
13. `pcb_t *outChild(pcb_t *p)`: make the PCB pointed to by `p` no longer the child of its parent. If the PCB pointed to by `p` has no parent, return NULL; otherwise, return `p`. Note that the element pointed to by `p` could be in an arbitrary position (i.e. not be the first child of its parent).

The Active Semaphore List (ASL)

- A *semaphore* is an important operating system concept
- For the purpose of this level it is sufficient to think of a semaphore as an integer
- Associated with this integer is:
  - A pointer to physical address in memory where the integer is stored
  - A process queue
- A semaphore is *active* if there is at least one PCB on the process queue associated with it

```
typedef struct semd_t {
    /* Semaphore key */
    int *s_key;
    /* Queue of PCBs blocked on the semaphore */
    struct list_head s_procq;

    /* Semaphore list */
    struct list_head s_link;
} semd_t, *semd_PTR;
```

Semaphores allocation

Main data structures:

- `semdFree_h`: free or not used semaphores' list.
- `semd_table[MAXPROC]`: array of semaphores with max size of `MAXPROC`.

Semaphore's functions to implement

1. `int insertBlocked(int *semAdd, pcb_t *p)`: insert the PCB pointed to by `p` at the tail of the process queue associated with the semaphore whose key is `semAdd` and set the semaphore address of `p` to semaphore with `semAdd`. If the semaphore is currently not active, allocate a new descriptor from the semdFree list, insert it in the ASL (at the appropriate position), initialize all of the fields, and proceed as above. If a new semaphore descriptor needs to be allocated and the semdFree list is empty, return TRUE. In all other cases return FALSE.
2. `pcb t *removeBlocked(int *semAdd)`: search the ASL for a descriptor of this semaphore. If none is found, return NULL; otherwise, remove the first PCB from the process queue of the found semaphore descriptor and return a pointer to it. If the process queue for this semaphore becomes empty, remove the semaphore descriptor from the ASL and return it to the semdFree list.

Semaphore's functions to implement

3. `pcb t *outBlocked(pcb_t *p)`: remove the PCB pointed to by p from the process queue associated with `p`'s semaphore on the ASL. If PCB pointed to by `p` does not appear in the process queue associated with `p`'s semaphore, which is an error condition, return NULL; otherwise, return `p`.
4. `pcb t *headBlocked(int *semAdd)`: return a pointer to the PCB that is at the head of the process queue associated with the semaphore `semAdd`. Return NULL if `semAdd` is not found on the ASL or if the process queue associated with `semAdd` is empty.
5. `initASL()`: initialize the semdFree list to contain all the elements of the array `static semd_t semdTable[MAXPROC]`. This method will be only called once during data structure initialization.

Suggestions

- There isn't a single way to implement Phase 1's functions
- Use `static` methods and variables where possible
- Learn to use a debugger: the embedded one or `gdb`
- Get familiar with `uriscv`

Alternative tricks for debugging in `uriscv`

- The provided file `klog.c` allows to “print” text and vars. On your code use the `klog_print` function, then on `uriscv` add the `klog_buffer` as a traced region.
- Create an empty function to simulate a break-point

```c
#include "klog.c"

void bp() {}

void scheduler() {
    ...
    klog_print("Sono arrivato qua\n");
    bp();
    ...
}
```

Setup tools

You need to install μRISCV:
`github.com/virtualsquare/uriscv/releases/latest`

This will also install the cross-compiler toolchain package:
`gcc-riscv64-unknown-elf`

Install `cmake` to compile the project

Provided files

These file are provided in a **.tar.gz**:

- `const.h`: constant (`MAXPROC`, etc...) definition file
- `type.h`: struct types (`pcb_t`, `semd_t`, etc...) definition file
- `listx.h`: functions and macros for list manipulation
- `pcb.h`: PCB header file
- `asl.h`: semaphore header file
- `klog.c`: print buffer helper
- `pcb.c`: **write your implementation here**
- `asl.c`: **write your implementation here**
- `CMakeLists.txt`: file to compile your project
- `p1test.c`: test file

Submission

The deadline is set for **Tuesday January 6, 2026 at 23:59**.

Upload a single `phase1.tar.gz` in the folder associated to your group with:

- All the source code with a `CMakeLists.txt`
- Documentation
- `README` and `AUTHOR` files

Please comment your code!

We will send you an email with the hash of the archive, you should check that everything is correct.

You will receive another email with the score out of 10.