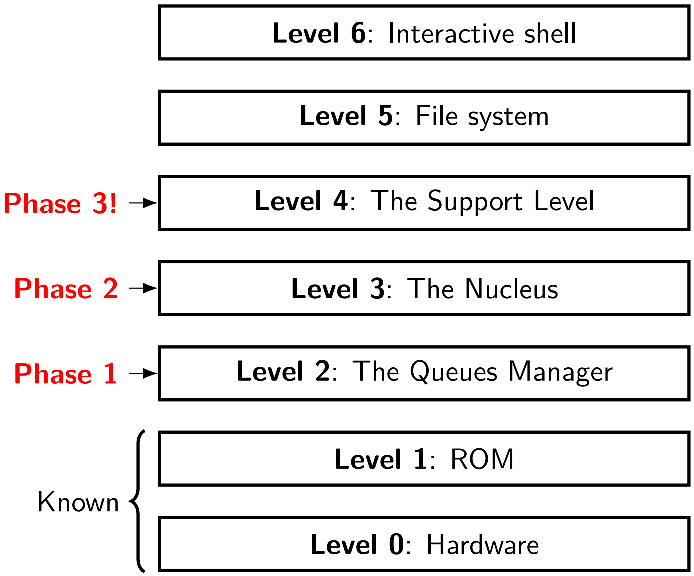
Kernel and μPandOS: Phase 3

Luca Bassi (luca.bassi14@studio.unibo.it)
Gabriele Genovese (gabriele.genovese2@studio.unibo.it)

April 16, 2024

μPandOS: six levels of abstraction



μPandOS: Phase 3 - Level 4: The Support Level

Level 4, the Support Level, builds on the Nucleus in two key ways to create an environment for the execution of user-processes:

1. Support for address translation/virtual memory.
2. Support for character-oriented I/O devices: terminals and printers.

```
typedef struct support_t {
    int         sup_asid;
    state_t     sup_exceptState[2];
    context_t   sup_exceptContext[2];
    pteEntry_t sup_privatePgTbl[USERPGTBLSIZE];
} support_t;
```

Goal of this phase

- You will be provided with 8 programs. This programs can be compiled and the 8 executables will be seen as as flash devices.
- You will write the `test` function that should upload the executables and the 8 programs should successfully execute.
- In addition to the scheduling you already have implemented, virtual memory and user services (SST) should be implemented to have an easier interaction with devices.

Important: This slides should not be used as a substitution to the specs. The following is a panoramic view of the specs.

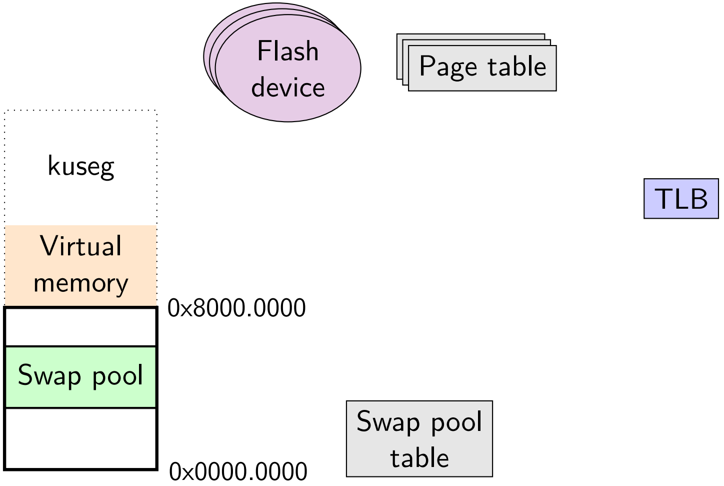
μPandOS virtual memory

The U-procs are executed in virtual memory (kuseg) that starts at 0x8000.0000.

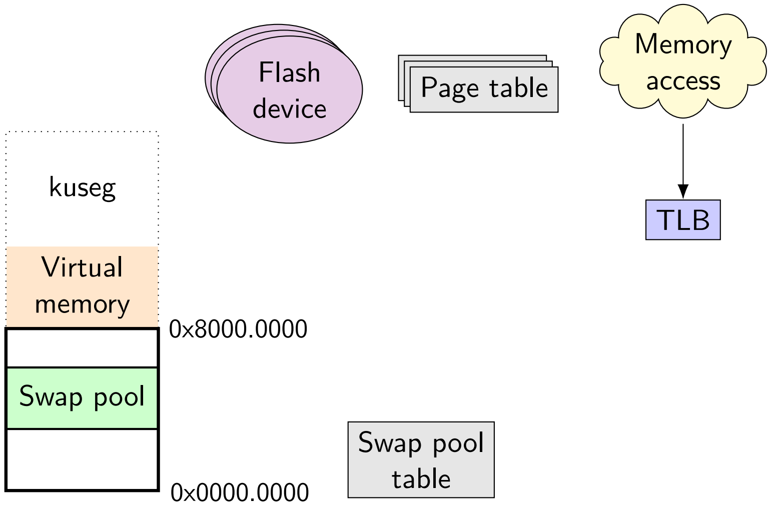
μPandOS uses a Translation Lookaside Buffer (TLB) to search for the corresponding physical address.

	EntryHI		EntryLo				
	VPN	ASID	PFN	N	D	V	G
0	0x80000	*i*			1	0	
1	0x80001	*i*			1	0	
⋮	⋮	⋮			⋮	⋮	
30	0x8001E	*i*			1	0	
31	0xBFFFF	*i*			1	0	

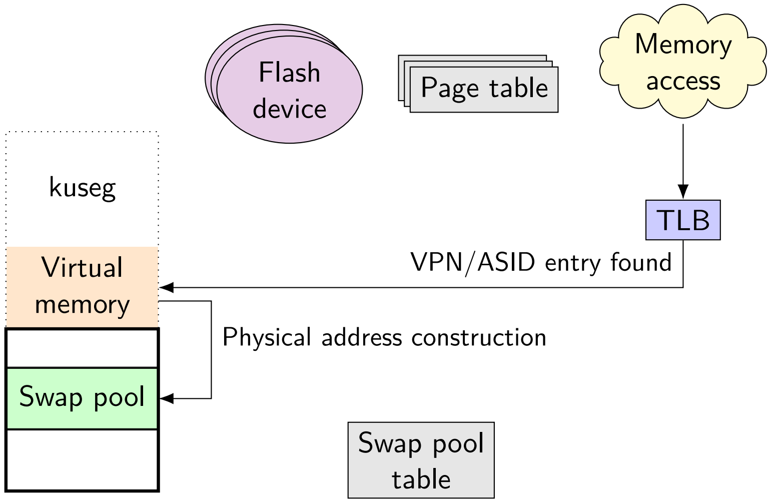
μPandOS virtual memory



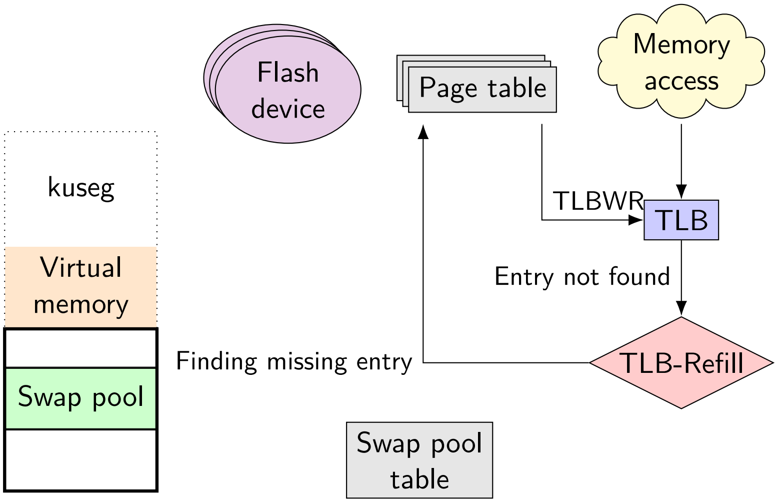
μPandOS virtual memory



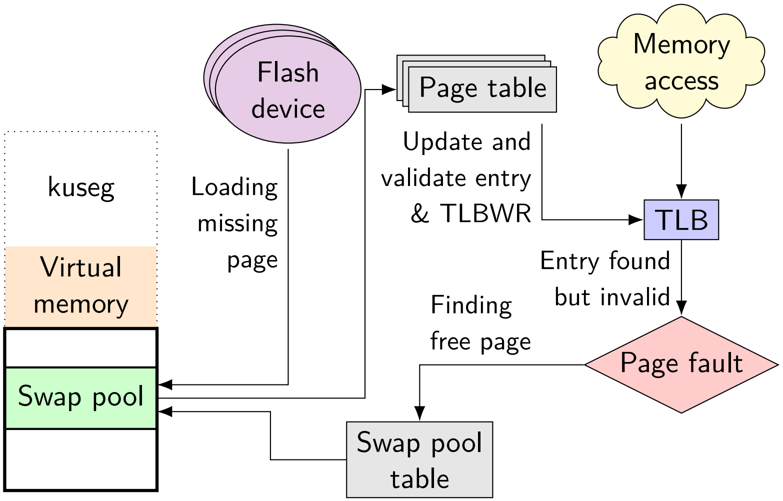
μPandOS virtual memory



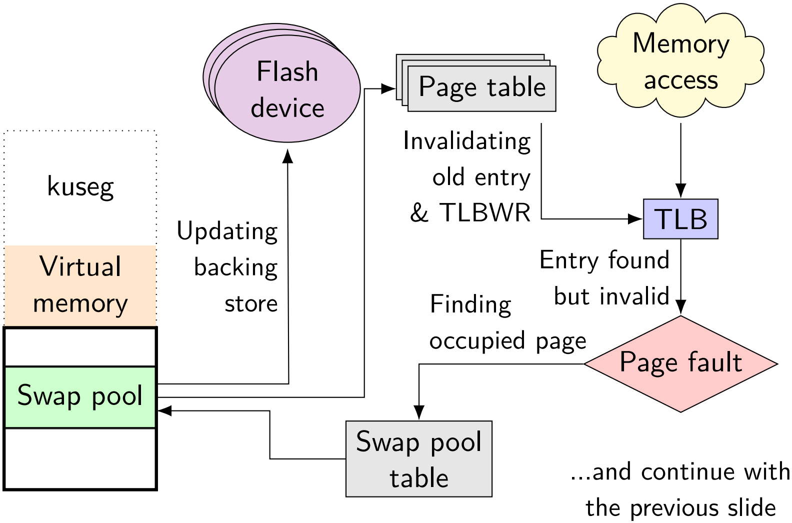
μPandOS virtual memory



μPandOS virtual memory



μPandOS virtual memory



Process initialization

The test process must initialize:

- Phase 3/Level 4's data structures
- The Swap Mutex process
- 8 System Service Thread (SST) processes
- Optionally, a process for each I/O device

After it must wait the 8 messages communicating the termination of U-procs.

There's no need to modify Phase 2/Level 3, because Phase 3/Level 4 builds on top of it. Obviously, you can modify the code of the previous phases to adapt your needs.

System Service Thread (SST)

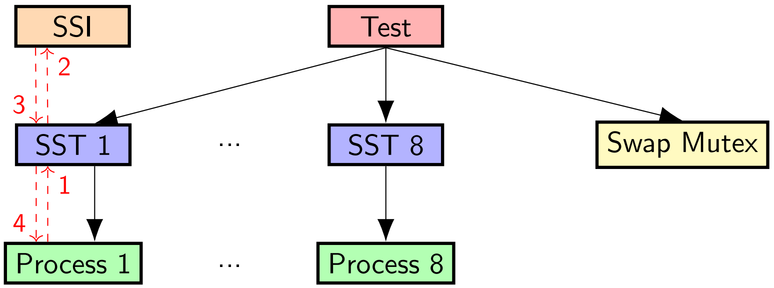
The System Service Thread (SST) is a per-process thread that provide is child process useful services. Each SST child process can send a message to its SST to request a service.

Each SST initialize it's child process that execute one of the testers.

Each SST must share the same ID (ASID) and support struct of its child U-proc.

The SST should initialize and start its child process.

Processes schema



SST Services

GetTOD	This service should allow the sender to get back the number of microseconds since the system was last booted/reset.
Terminate	This service causes the sender U-proc and its SST (its parent) to cease to exist. It is essentially a SST “wrapper” for the SSI service TerminateProcess. Remember to send a message to the test process to communicate the termination of the SST.
WritePrinter	This service cause the print of a string of characters to the printer with the same number of the sender ASID.
WriteTerminal	This service cause the print of a string of characters to the terminal with the same number of the sender ASID.

Syscall: SendMsg (USYS1)

This services cause the transmission of a message to a specified process. The USYS1 service is essentially a user-mode “wrapper” for the kernel-mode restricted SYS1 service.

If `a1` contains `PARENT`, then the requesting process send the message to its SST, that is its parent.

Syscall: ReceiveMsg (USYS2)

This system call is used by a process to extract a message from its inbox or, if this one is empty, to wait for a message. The USYS2 service is essentially a user-mode “wrapper” for the kernel-mode restricted SYS2 service.

Testing

There is a provided set of possible U-proc programs that will “exercise” your code. These programs will generate page faults in addition to issuing SYSCALLs 1-2 and purposefully causing Program Traps.

The supplied U-proc programs also come with their own `Makefile` configured to compile, link (using the U-proc linker script, `crtsi.o`), create a corresponding flash device (a `.umps` file), and preload the U-proc’s load image on to a flash device.

Submission

The deadline is set for **Sunday June 9, 2024 at 23:59** or **Sunday July 14, 2024 at 23:59** or **Sunday September 8, 2024 at 23:59**.

Upload a single `phase3.tar.gz` in the folder associated to your group with:

- All the source code with a Makefile
- 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.