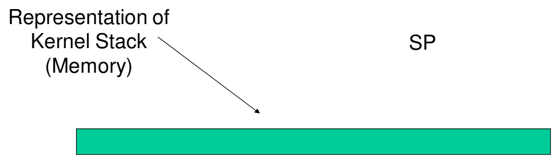
Assignment 2

Fork

- Proc_fork (*)
  - Proc create
  - As_copy()
  - Pid_allocation
  - File related clone of parent
  - Current working directory

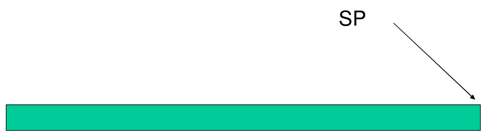
Fork & Trapframe

- Running in user mode, SP points to user-level stack (not shown on slide)



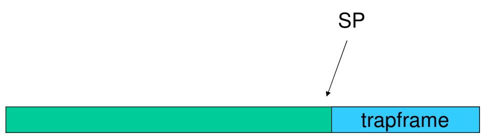
Example Context Switch

- Fork syscall and we switch to the kernel stack



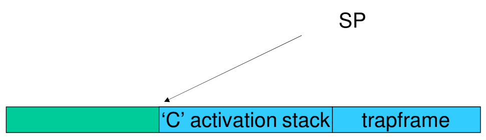
Example Context Switch

- We push a *trapframe* on the stack
  - Also called *exception frame, user-level context….*
  - Includes the user-level PC and SP



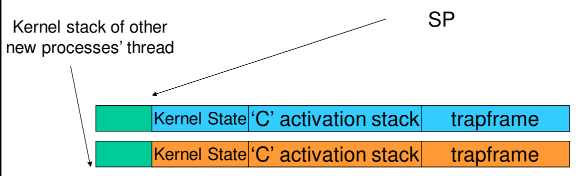
Example Context Switch

- Call ‘C’ code to process syscall
- Results in a ‘C’ activation stack building up



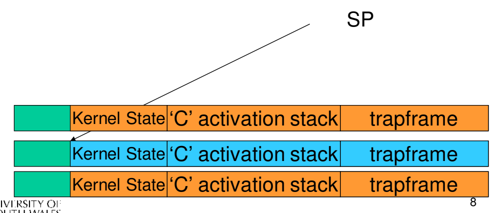
Fork

- Creates a thread must
  - and have a similar stack layout to the stack we are currently using
  - Trapframe need to be on the stack
    - Does not matter where (local variable)
  - Enter_forked_process
    - Tweaks trapframe prior to calling md_usermode



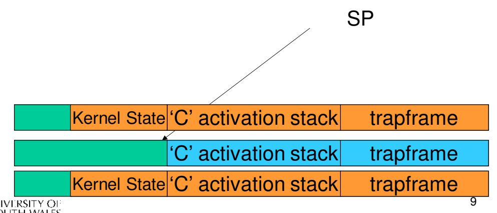
Context Switch

- We save the current SP in the PCB (or TCB), and load the SP of the target thread.
  - Thus we have *switched contexts*



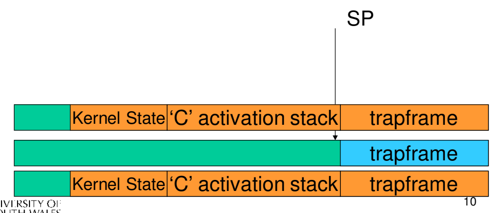
Example Context Switch

- Load the target thread's context, and return to C
  - Enter_forked_process



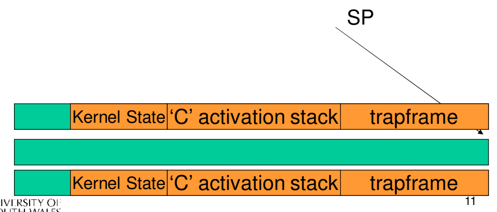
Example Context Switch

- The C continues and (in this example) returns to user mode.
  - mips_usermode



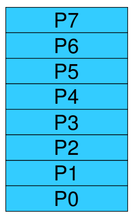
Example Context Switch

- The user-level context is now child



Pids of Processes

- A processes' information is stored in a *process control block* (PCB)
  - Reality is much more complex (hashing, chaining, allocation bitmaps,…)
- Two main parts
  - Allocate unused pid
  - Look up a struct proc given a pid



waitpid

- See the man page
- Scenarios:
  - Child exit after waitpid
  - Child exits before waitpid
  - Parent exits before child exits
    - waitpid never called?
- Tradition unix behaviour:
  - https://en.wikipedia.org/wiki/Zombie_process

Execv

- Create a new address space
- Destroy the old
  - as_activate() the new
- Copying the arguments to the child
  - Copy into kernel
  - Copy out into child
- Similar to run program

Execv - args

- A null terminated array of pointers to strings
  - Passing into parent
  - Expected to be in child
  - See Note: userland/lib/crt0/mips/crt0.S
    - Register A0 = argc
    - Register A1 = argv
- Note: You can assume ARG_MAX = 4K