OS Design 2017-2018

POSIX context management and control

Task context

- Minimum set of task data that needs to be saved to allow a task to be interrupted and later restored
- Example
  - Registers
  - Memory (stack)

POSIX context structure

```
typedef struct ucontext {

    struct ucontext uc_link; /* context that will be resumed when the current context
terminates*

    sigset_t        uc_sigmask; /* set of  signals  blocked  in  this  context */

    stack_t         uc_stack; /* stack  used  by this context */

    mcontext_t      uc_mcontext; /* machine-specific  representation  of

   the  saved  context */

    ...

} ucontext_t;
```

getcontext

- int getcontext(ucontext_t *ucp);
- initializes the structure pointed at by ucp to the currently active context.
- Why:
  - When needed to save the current context for later use
  - Incrementally build a new context from the current one

setcontext

- int setcontext(const ucontext_t *ucp)
- Restores the user context pointed at by ucp
- A successful call does not return

- When to use:
  - When switching to an already created context

makecontext

- void makecontext(ucontext_t *ucp, void *func(), int argc, ...)
- Creates an alternate thread of control in ucp, which has previously been initialized using getcontext
- ucp.uc_stack must be allocated
- When using set context or swapcontext the execution begins at entry point of func
- When func terminates, control is returned to ucp.uc_link

swapcontext

- int swapcontext(ucontext_t *oucp, ucontext_t *ucp)
- Transfers control to ucp and saves the current execution state into oucp