Operating Systems

User Contexts

• A user context (\texttt{ucontext}) defines the minimal state of a thread – the minimal set of data that must be saved to allow it to be interrupted
• These are not the same as “threads”.
• Threads are built on top of \texttt{ucontext}s.
• A thread shares its state with the underlying process except for the machine context.
• The major task for a user-space threading system is to create and dispatch those machine contexts.

User Contexts

Saving State

• The idea of a context is to save the state of the machine/program/process in such a way that it can be restored later.
  – Signal Mask
  – Stack
  – Machine Context
  – link to next context
  – …

Saving State

Signal Mask

• The set of signals blocked when this context is used.
• Threads can register independent signal handlers and signal masks.

Signal Mask

Stack

• Imagine the confusion if multiple threads were using the same stack.
• Saving the stack (and having an independent one for each \texttt{ucontext}) allows for function calls from the start function of a context.

Stack

Machine State (Context)

• CPU Stores State in Registers
  – Integer Registers
  – Floating Point
• \texttt{mcontext_t} stores them.

Machine State (Context)
Using Contexts

```c
using_t uc;
void *stack;

getcontext(&uc); //initialize with current state
stack = malloc(SS); // create new stack
/* configure fields */
uc.uc_stack.ss_sp = stack;
uc.uc_stack.ss_size = SS;
uc.uc_stack.ss_flags = SS_DISABLE;
sigemptyset(&uc.uc_sigmask); //configure sigmask
//set initial function and arguments
makecontext(&uc, function, 0);
```

Passing Arguments

```c
void f_add(int x, int y) {output(x+y);} 
makecontext(uc, f_add, 2, 7, 20);
```

Context Switching

```

```

Timing the Switch

- In our setup, each of these dots represents a timer tick.
- Our program receives a SIGALRM.
- We use an Interval Alarm (timer) for this.
- The scheduler is called from the signal handler routine.