CS202

1. Attaching to a Debugger

1.1 Launch a process that is attached:

```
void launch_attached(const char* path,
        char* const argv[]) {
  int pid = fork();
  if (pid == 0) {
    ptrace(PTRACE_TRACEME, 0, NULL, NULL);
    execv(path, argv);
  }
  return pid;
}
```

1.2 Attach to a running process

```
void attach_to_process(pid_t pid) {
    ptrace(PTRACE_ATTACH, pid, NULL, NULL);
}
```

1.3 Continue execution once attached

```
void continue_once_attached(pid_t pid) {
  while (1) {
    int status;
    waitpid(pid, &status);
    if (WIFSTOPPED(status)) {
      // The reason for the change
      // was that pid stopped.

      // We should have stopped because of
      // either SIGTRAP and SIGSTOP.
      assert(WSTOPSIG(status) == SIGTRAP
        || WSTOPSIG(status) == SIGSTOP);

      // Continue execution
      ptrace(PTRACE_CONT, pid, NULL, NULL);
      break;
    } else if (WIFEXITED(status)) {
      // The process exited before we could
      // attach.
      printf("Process exited\n");
      break;
    }
  }
}
```

2. Interrupting the running thread

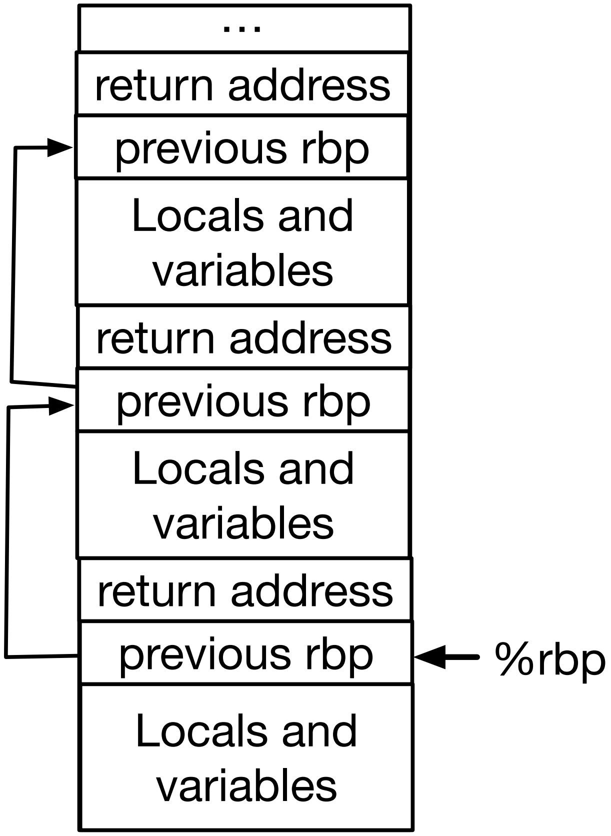
```c
void interrupt(pid_t pid) {
    kill(pid, SIGSTOP);
    // Must use waitpid in order to
    // wait for the signal to be
    // delivered.
}
```

3. Other ptrace commands

All of these only make sense when the target process is stopped, for instance due to the use of `interrupt` from above.

```c
// Execute a single instruction in the process.
ptrace(PTRACE_SINGLESTEP, pid, NULL, NULL);

// Get non-floating point registers.
// This includes rsp, rip, rbp, etc.
struct user_regs_struct regs;
ptrace(PTRACE_GETREGS, pid, &regs, NULL);

// Get floating point registers.
struct user_fpregs_struct fpregs;
ptrace(PTRACE_GETFPREGS, pid, &fpregs, NULL);

// Set registers. This can be used to update
// register values.
ptrace(PTRACE_SETREGS, pid, &regs, NULL);

// Note: PTRACE_PEEKUSER and PTRACE_POKEUSER
// provide a more efficient way to read or
// write a single register.

// Read a word (8 bytes) from address `addr`
// in target process memory. Note, despite being
// called PTRACE_PEEKDATA, on Linux this can
// read any part of memory, including the
// text segment.
uint64_t val;
val = ptrace(PTRACE_PEEKDATA, pid, addr, NULL);

// Write a word (8 byte) to address `addr` in
// target procss memory.
ptrace(PTRACE_POKEDATA, pid, addr, val);

// Get information on the signal that caused
// the target procss to stop.
siginfo_t sinfo;
ptrace(PTRACE_GETSIGINFO, pid, &sinfo, NULL);
```

4. **Stack Frames and Unwinding**


...
return address
previous rbp
Locals and variables
return address
previous rbp
Locals and variables
return address
previous rbp
%rbp
Locals and variables