Processes Environment -- Chapter 7

Program execution: (not new process!)

- int main (argc, char *argv[], char *envp[]);

Process termination

- exit(3) and _exit(2)
  - zero value
  - non-zero value
- Return from main(), exit(main(...))
- abort(3) -- (later ...)

- atexit(3)
  - schedule a function to be run at exit() time.
  - called in reverse order of registration.
  - _exit() does not call registered functions.

Memory Layout of a C program

Segments: text, initialized data, uninitialized data (bss), stack

High Addresses

- system
- command line args, env vars
- stack
- heap
- uninitialized data
- initialized data
- text
- unmapped (4k)

Low Addresses

size(1)

Shared Libraries

Memory Allocation

Heap based

- malloc(3)
- calloc(3) -- initializes to zero
- realloc(3) -- changes a size of a previously allocated block
- alloca(3) -- discouraged, very machine dependent

- free(3)

System calls:

- brk(2), sbrk(2) -- changes data segment size

setjmp & longjmp

goto statement in C ...

- int setjmp(jmp_buf env);
- void longjmp(jmp_buf env, int val);

- setjmp()
  - returns 0 on direct call
  - returns non-zero (val) from longjmp

- typical use -- errors deep in call sequence

Problems with Automatic variables

What is an automatic variable?

```
 int *ipf()
 { int j;
   j = 35;
   return &j;
 }
```

- j is an automatic variable, allocated on the stack
- after return, stack is "reused"
  - improper use of an automatic variable, big problem
- static variables, ok.
  - static int j;

Resource Limits

```
int getrlimit(int resource, struct rlimit *rlp);
int setrlimit(int resource, const struct rlimit *rlp);
```

- RLIMIT_*
  - _CORE, _CPU, _DATA, _FSIZE, _MEMLOCK
  - _NOFILE, _NPROC, _RSS, _STACK

- soft limit vs hard limit

- Shell access:
  - csh - limit
  - sh/bash/ksh - ulimit
    - hard vs soft