Chapter 7: The Environment of a Unix Process

CMPS 105: Systems Programming
Prof. Scott Brandt
T Th 2-3:45
Soc Sci 2, Rm. 167

The main() function

- int main(int *argc*, char **argv*[]);
- *argc* = number of command-line arguments
- *argv* = array of pointers to the (string) arguments
- main() is the first thing called in the program
- A special start-up routine is called first (specified in the executable)
  - That’s what sets up the parameters to main

Process Termination

- Five ways to terminate a process
- Normal termination
  - return from main()
  - call exit()
  - call _exit()
- Abnormal termination
  - call abort()
  - terminate by a signal

exit() and _exit()

- #include <stdlib.h> (ANSI C)
- void exit(int *status*);
  - Performs a clean shutdown of the standard I/O library
- #include <unistd.h> (POSIX)
- void _exit(int *status*);
- Exit status undefined if not specified

atexit()

- ANSI C: A process can register up to 32 *handler* functions to execute when the program exits
    - Typically used to clean up
- #include <stdlib.h>
- int atexit(void (**func*)(void));
- *func* is a pointer to a function that takes no parameters
    - Specified by using the name of the function (without parantheses)

Exit handling

- Draw and discuss Figure 7.1 on page 164

Command-line arguments

- Programs can pass command-line parameters

```
#include <ourhdr.h>
int main(int argc, char *argv[]) {
   int i;
   for(i=0; i < argc; i++)
         print(“argv[%d]: %s\n”, i, argv[i]);
   exit(0);
}
```

Environment List

- Each program is passed an environment list
- extern char **environ;
- Each environment string consists of name=value
- Most names are uppercase
- Usually ignored, but can be useful
  - Why?

Memory Layout of a C Program

- Text segment
  - The machine instructions of the program
  - Usually sharable and read-only
- Data segment (initialized data)
  - Global variables that are initialized in the program
- BSS (uninitialized data)
  - Global variables that are not initialized in the program
  - Initialized to zero or null pointers
- Stack (automatic variables)
  - Function return information
  - Local variables
- Heap
  - Dynamic memory allocation
- See figure 7.3 on page 168

Shared Libraries

- Single shared copy of common library routines
  - Instead of each one being copied in each program
- Big space savings
  - 24576 vs. 104859 for hello world
  - For details, see comparison on page 169

Memory Allocation

- #include <stdlib.h>
- void *malloc(size_t *size*);
  - Allocates the specified number of bytes
  - Uninitialized
- void *calloc(size_t *nobj*, size_t *size*);
  - Allocates space for the specified number of objects
  - Initialized to all 0s
- void *realloc(void **ptr*, size_t *newsize*);
  - Changes the size of a previously allocated area
  - May move to a new location (and copy old contents)
  - New area is uninitialized
- void *free(void **ptr*);
  - Frees allocated space

Common mistakes

- Writing past the end of an allocated region or variable
  - Overwrites record-keeping information or other data
  - Really, really hard to find
- Failing to free memory
  - Memory leaks
  - Big problem when not using virtual memory
- Freeing memory more than once
  - May cause memory to be allocated twice!
- Calling free() with a bad pointer
  - Free tries to free up whatever is pointed to by the pointer
- Can be caught with special memory management functions
  - Not automatically checked because of overhead involved

alloca

- Allocates memory from the stack
- Doesn’t have to be freed
- Doesn’t live past the return from the calling function

Environment variables

- Used by applications only (not the kernel)
- name=value
- Common: HOME, USER, PRINTER, etc.
- #include <stdlib.h>
- char *getenv(const char **name*);
    - Returns null if not found

Other Environment functions

- int putenv(const char **str*);
  - Creates (or overwrites) environment variable
- int setenv(const char *name, const char *value, int rewrite);
  - Same as putenv (modulo params), except
  - Does nothing if rewrite = 0 and old value exists
- int unsetenv(const char *name);
  - Clears an environment variable

setjmp() and longjmp()

- Allow gotos from lower in a call stack to higher in a call stack
- setjmp sets up the location to jump to
- longjmp jumps there
- Parameter contains the environment of the function that will be jumped to
- Bottom line: don’t use these!

getrlimit and setrlimit

- Query and change resource limits
- #include <sys/time.h>
- #include <sys/resource.h>
- int getrlimit(int *resource*, struct rlimit **rlptr*);
- int setrlimit(int *resource*, const struct rlimit **rlptr*);

```
struct rlimit {
    rlim_t rlim_cur; /* soft limit: current limit */
    rlim_t rlim_max; /* hard limit: max value */
}
```

Resources

- RLIM_INFINITY = unbounded
- RLIMIT_CORE: max core file size (0 = none)
- RLIMIT_CPU: max CPU time in seconds
- RLIMIT_DATA: max size of data segment
- RLIMIT_FSIZE: max size in bytes of a file that may be created
- RLIMIT_MEMLOCK: locked-in memory space
- RLIMIT_NOFILE: max # open files
- RLIMIT_NPROC: max # of child processes per real user ID
- RLIMIT_OFILE: same as RLIMIT_NOFILE
- RLIMIT_RSS: max resident set size in bytes (max memory footprint)
- RLIMIT_STACK: max stack size
- RLIMIT_VMEM: max size of mapped address space (affects mmap)