COMP 2400 UNIX Tools

Christian Grothoff
christian@grothoff.org

http://grothoff.org/christian/
Time is Relative

- Timezones, DST
- Y2K
- February 29th, leap seconds
- US Congress
- Precision (s, ms, ns?)
Time

- `time_t time(time_t * tloc)`
- `int gettimeofday(struct timeval * tp, struct timezone * tz)`
- `struct tm * localtime(const time_t * clock)`
- `char * nl_langinfo(nl_item item) – D_FMT`
- `size_t strftime(char * s, size_t max, const char * format, const struct tm * tm)`
One Process, many Users

- Main user ID: who runs the process
- Effective user ID: permissions for the process
- Filesystem user ID: default owner of files created by the process
- Saved user ID: UID that the process used to have and is allowed to switch back to

Different UNIX variants differ! Check man pages!
User API

- `char * getlogin(void)` – not secure!
- `uid_t getuid(void)` – user executing
- `uid_t geteuid(void)` – user owning SUID executable
- `int setuid(uid_t uid)`
- `int seteuid(uid_t uid)`
Group API

- `gid_t getgid(void)`
- `gid_t getegid(void)`
- `int setgid(gid_t gid)`
- `int setegid(gid_t gid)`
- `int getgroups(int gssize, gid_t * grouplist)`
- `int setgroups(int ngroups, const gid_t * groups)`
Information about Users and Groups

- struct passwd * getpwnam(const char * name)
- struct passwd * getpwuid(uid_t uid)
- struct group * getgrnam(const char * name)
- struct group * getgrgid(gid_t gid)
Process Resource Limits

- int getrlimit(int resource, struct rlimit * rlp)
- int setrlimit(int resource, const struct rlimit * rlp)
- RLIMIT_VMEM: process’ address space (malloc + mmap)
Signals

• Signals are **software interrupts**

• Examples: illegal instruction, division by zero, segmentation violation, terminal closed, CTRL-C, etc.

• Possible actions: ignore, block (delay until unblocked), catch (call a signal handler) or die

• Not all actions are possible for all signals, each signal has a default action
Common Signals

- SIGHUP, SIGINT, SIGQUIT, SIGTERM, SIGABRT, SIGKILL
- SIGFPE, SIGILL, SIGBUS, SIGSEGV
- SIGTRAP, SIGPROF, SIGUSR1, SIGUSR2
- SIGPIPE, SIGALRM, SIGCHLD
- SIGSTOP, SIGCONT
Signal Handling

- `pid_t getpid()`
- `int kill(pid_t pid, int sig)`
- `int pause(void)` – usually `select` is better!
- `typedef void (*sighandler_t)(int)`
- `sighandler_t signal(int signum, sighandler_t handler)`
Modern Signal Handling

- `int sigaction(int signum, const struct sigaction * act, struct sigaction * old)`

  ```c
  struct sigaction {
      void (*sa_handler)(int)
      // ...
      int sa_flags;
  }
  ```
Funky Control Flow

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

>99.999% of the time it is a very bad idea to use these functions!
Process Termination

- `return from main method`
- `void exit(int status)`
- `void abort(void)`
- `void _exit(int status)`
- `int atexit(void (*function)(void))`
Zombies!

• Exit status of process must be communicated to parent
• Parent may not acknowledge status immediately

⇒ Zombie process is left

You cannot kill zombies, but you can kill their parents (if they fail to acknowledge)!
Init

• Process 1
• Parent of orphans
• Reads (and discards) exit status

⇒ Orphaned zombies die immediately
Be nice!

- int nice(int incr)
- Only root can have a negative priority
Executing other Programs

- int system(const char * string)
- int execvp(const char * file, const char * argv[])
- pid_t fork(void)
- pid_t wait(int * status)
- pid_t waitpid(pid_t pid, int* status, int options)
Threads

- `int pthread_create(pthread_t * thread, pthread_attr_t * attr, void * (*start)(void*), void * arg)`
- `int pthread_join(pthread_t th, void ** retval)`
- `int pthread_mutex_init(pthread_mutex_t * mutex, const pthread_mutexattr_t * mutexattr)`
- `int pthread_mutex_lock(pthread_mutex * mutex)`
- `int pthread_mutex_unlock(pthread_mutex * mutex)`
- `int pthread_mutex_destroy(pthread_mutex * mutex)`
More APIs

• Condition variables, semaphores
• Interprocess communication (IPC)
• Sockets (networking!)
• Asynchronous IO
• Command line option parsing (getopt)

Learning SVR4 is a life-long process!
Questions