Operating Systems

Project 2 Feedback

Shell Signal Handling

• ctrl-c sends SIGINT
• ctrl-d sends EOF, not a signal
• SIGINT, SIGTERM, SIGTTIN, SIGTTOU, SIGTSTP, SIGQUIT
  – Shell should not be interruptible/stoppable
  – Child processes should be
• Children inherit parent’s signal masks
  – If the shell ignores ctrl-c, so does its children
  – Set the signals back to default before you replace the child process

Shell behavior

• stick to Unix responses
  – %s: Command not found
  – Type “exit” to exit
• “cd” doesn’t work in your shell
• redirections/pipes
  – your shell must parse and implement them
  – feeding the symbols as arguments to commands is not the way

Check errno

• execvp fails for many reasons
  – E2BIG, EACCES, EAGAIN, EFAULT, EINTR, ELOOP, ENAMETOOLONG, ENOENT, ENOLINK, ENOTDIR, ENOEXEC, ENOMEM, ETXTBSY
  – “Command not found” should only be printed in response to ENOENT

Valgrind

• “Still reachable” must be freed
  – it means that you haven’t lost the pointer(s) to the heap memory, but you haven’t freed it either
• Memory errors
  – Invalid reads/writes
  – conditional jump or move depends on uninitialised values(s)
  – param points to uninitialised bytes
• Use flags
  – --leak-check=full
  – --show-leak-kinds=all

Integer Literals

• EXIT_SUCCESS/EXIT_FAILURE // stdlib.h
• Command history size
• % constant
Tokenizer

typedef struct tokenizer {
    char *str; // the string to parse
    char *pos; // position in string
} TOKENIZER;

char *get_next_token(TOKENIZER *tokenizer);
//if current char is a delimiter, just return it
//else go until next char is a delimiter
//return the substring without white spaces
//return NULL when string ends

How the Tokenizer is used

// return # of tokens
// store pointers to tokens
int parse() {
    int i = n = 0;
    line = readline(PROMPT);
    if(line == NULL)
        return 0;
    //newline
    if(strcmp(line,"") == 0)
        return 0;
    add_history(line);
    t = init_tokenizer(line);
    while(get_next_token(t) != NULL)
        n++;
    // allocate pointers to tokens
    // +1 for the ending NULL
    toks = (char**) malloc(sizeof(char*)*(n+1));
    // start from the beginning again
    t = init_tokenizer(line);
    while((toks[i++] = get_next_token(t)) != NULL);
    return n;
}

PRINTF Formatting

• Precision %.#f
• Width %#f, %#d
  – Note: Entire width
• Zero-padding %0#d
• Left-justification %-#d
• Various combinations of the above

Formatting Example (1)

%f with 1.23456789 >1.234568<
%.10f with 1.23456789 >1.2345678900<
%.2f with 1.23456789 >1.23<
%d with 12345 >12345<
%10d with 12345 >     12345<
%-10d with 12345 >12345<
%f with 1.23456789 >1.234568<
%8.2f with 1.23456789 >1.23<

Formatting Example (2)

%d:%d with 1 and 5 >1:5<
%02d:%02d with 1 and 5 >01:05<
%10d with 12345 >     12345<
%-10d with 12345 >12345<