PURDUE UNIVERSITY®

CS 25200: Systems Programming

Lecture 15: Signals, Built-ins, and Subshells

Prof. Turkstra

Evaluations

- Mid-semester course evaluations available
- Some notes on third party “evaluations”

2b - University questions about your instructor		Turkstra, Jeffrey A								
		Responses					Individual			
		E	G	F	P	VP	N	Grp Med	Mode	Std Dev
Q6	Overall, I would rate this instructor as:	17	36	13	7	4	77	3.9	4	1.07

Responses: [E] Excellent=5 [G] Good=4 [F] Fair=3 [P] Poor=2 [VP] Very Poor=1

3c - PICES instructor based questions		Turkstra, Jeffrey A								
		Responses					Individual			
		SA	A	U	D	SD	N	Grp Med	Mode	Std Dev
Q19	My instructor takes my views and comments seriously.	21	31	18	3	1	74	4.0	4	.90

Responses: [SA] Strongly Agree=5 [A] Agree=4 [U] Undecided=3 [D] Disagree=2 [SD] Strongly Disagree=1

CS standard questions about instructor		Turkstra, Jeffrey A								
		Responses					Individual			
		SA	A	U	D	SD	N	Grp Med	Mode	Std Dev
Q20	My instructor seemed concerned that students learn.	21	35	13	4	1	74	4.0	4	.89
Q21	My instructor showed a clear understanding of the subject.	45	22	5	0	2	74	4.7	5	.84
Q22	My instructor was well-prepared and organized in class.	30	32	11	1	0	74	4.3	4	.75
Q23	My instructor helped me understand the material.	13	32	20	7	2	74	3.8	4	.97
Q24	My instructor gave exams that accurately assessed what I have learned in this course.	14	28	13	13	6	74	3.7	4	1.21
Q25	My instructor graded fairly.	15	33	17	8	0	73	3.8	4	.90
Q26	My instructor was reasonably available to help students outside of class.	20	35	14	3	0	72	4.0	4	.80
Q27	My instructor encouraged in-class participation.	25	36	10	3	0	74	4.2	4	.79
Q28	My instructor shows respect for me and other students in this class.	23	35	12	3	1	74	4.1	4	.87
Q29	My instructor is open to student's questions.	36	33	4	1	0	74	4.5	5	.66
Q30	My instructor encourages an atmosphere where ideas can be exchanged freely and easily.	30	34	7	2	1	74	4.3	4	.83

Responses: [SA] Strongly Agree=5 [A] Agree=4 [U] Undecided=3 [D] Disagree=2 [SD] Strongly Disagree=1

Grades

- Quiz grades updated
  - Quiz 9 grade
- Lab attendance up soon
- Remember, regrade deadline is one week

Project 1 Stats

- Average: 105.5/125 (84.4%)
- Standard Deviation: 22.7/125 (18.2%)
- Median: 114.5/125 (91.6%)

Project 2 Stats

- Average: 78.7
- Standard Deviation: 28.9
- Median: 91.6

* excluding students that did not submit

Lecture 15

- Signals
- Built-ins
- Subshells

Signals

- One form of inter-process communication (IPC)
- Asynchronous mechanism for the OS to communicate with a running process
- Processes can register signal handlers to perform certain actions for certain signals
- Signals are similar to interrupts

Some signals

- SIGHUP: Hangup
- SIGINT: Terminal interrupt
- SIGBUS: BUS error
- SIGKILL: Kill (cannot be ignored)
- SIGSEGV: Segmentation violation
- SIGTERM: Termination
- SIGCHLD: Child process has stopped, exited, or changed
- SIGUSR1, SIGUSR2, etc

Handling signals

- sighandler_t signal(int signum,
  sighandler_t handler)
- int sigaction(int signum,
  const struct sigaction *act,
  struct sigaction *oldact);

```
struct sigaction {
        void     (*sa_handler)(int);
        void     (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t   sa_mask;
        int        sa_flags;
        void     (*sa_restorer)(void);
    };
```

Flags

- SA_RESTART: Resume the function after a signal is handled properly
- Instead of returning EINTR
- SA_NOCLDSTOP: Only deliver SIGCHLD on termination, not stopping
- SA_ONSTACK: Use the signal stack
  - Must set it up first

Signals and lex

- Lex's scanner uses getc() to read from fd 0
- getc() is built on top of the read() system call
- Many blocking system calls will return if a signal is received
  - And set errno to EINTR
- What happens when we get SIGINT (or SIGCHLD)?
  - getc() returns -1!
- How do we stop it?

Keeping lex alive

- ...use SA_RESTART

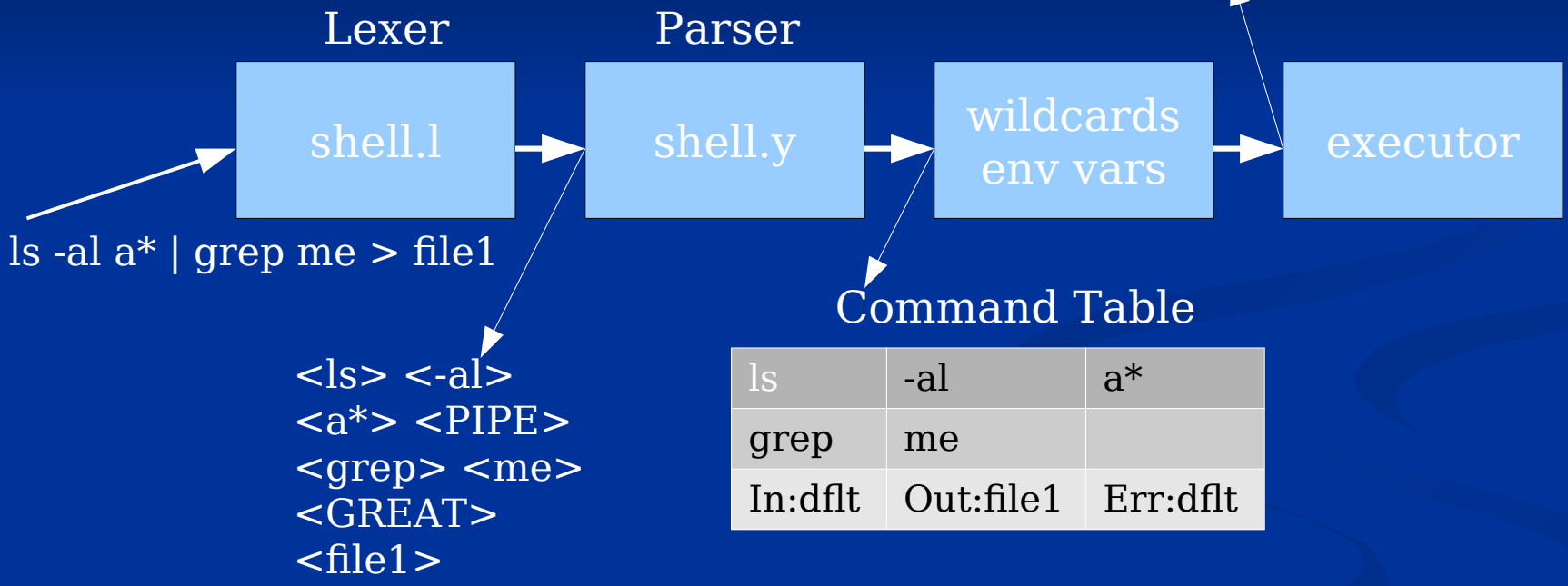
```
struct sigaction signal_action;

signal_action.sa_handler = sig_int_handler;
sigemptyset(&signal_action.sa_mask);
signal_action.sa_flags = SA_RESTART;
int error = sigaction(SIGINT,
                      &signal_action, NULL);
if (error) {
  perror("sigaction");
  exit(-1);
}
```

Shell

Final Command Table

ls	-al	aab	aaa
grep	me		
In:dflt	Out:file1	Err:dflt	



Built-ins

- Some “commands” are not discrete programs
- Usually simple ways to alter the shell’s state
  - E.g. cd, setenv, unsetenv, printenv, etc

Built-ins

- Should we fork?
- cd: See the chdir() system call
- setenv: Set an environment variable
- unsetenv: Unset it
- printenv: Display environment variables
- source: Execute each line as though it were in the input stream

printenv

- Linux implements printenv as a program :-)
- But…

```
ret = fork();
if (ret == 0) {
  if (!strcmp(argument[0], ”printenv”)) {
    char **p = environ;
    while (*p != NULL) {
      printf(“%s”,*p++);
    }
    exit(0);
  }
  …
  execvp(...);
```

Velociraptors

- "It's a UNIX system! ... I know this!"

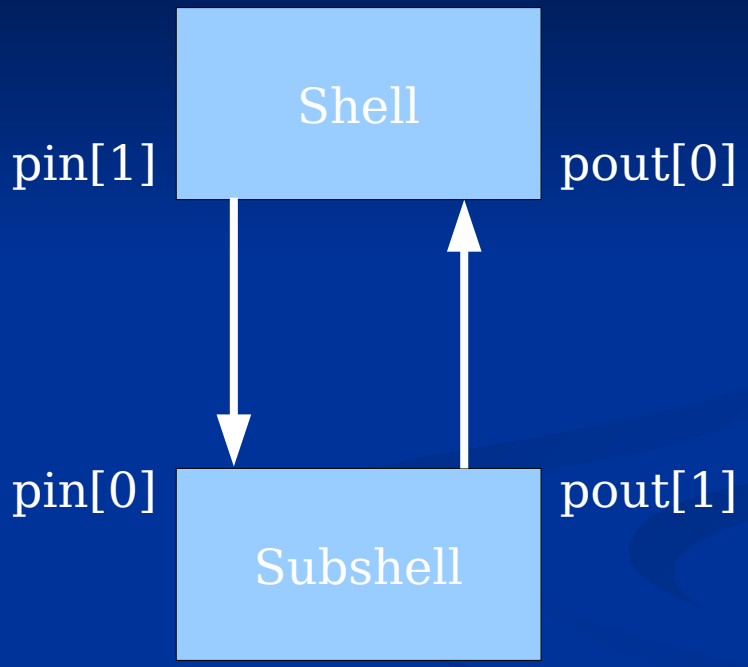
Subshell

- Subshells allow commands to execute in a shell environment without impacting the parent shell
  - Copy of output is returned to the parent shell
  - Parent shell fork()s and exec()s itself
  - Requires bidirectional communication

Two pipes

```
int pin[2];
int pout[2];

pipe(pin);
pipe(pout);
```



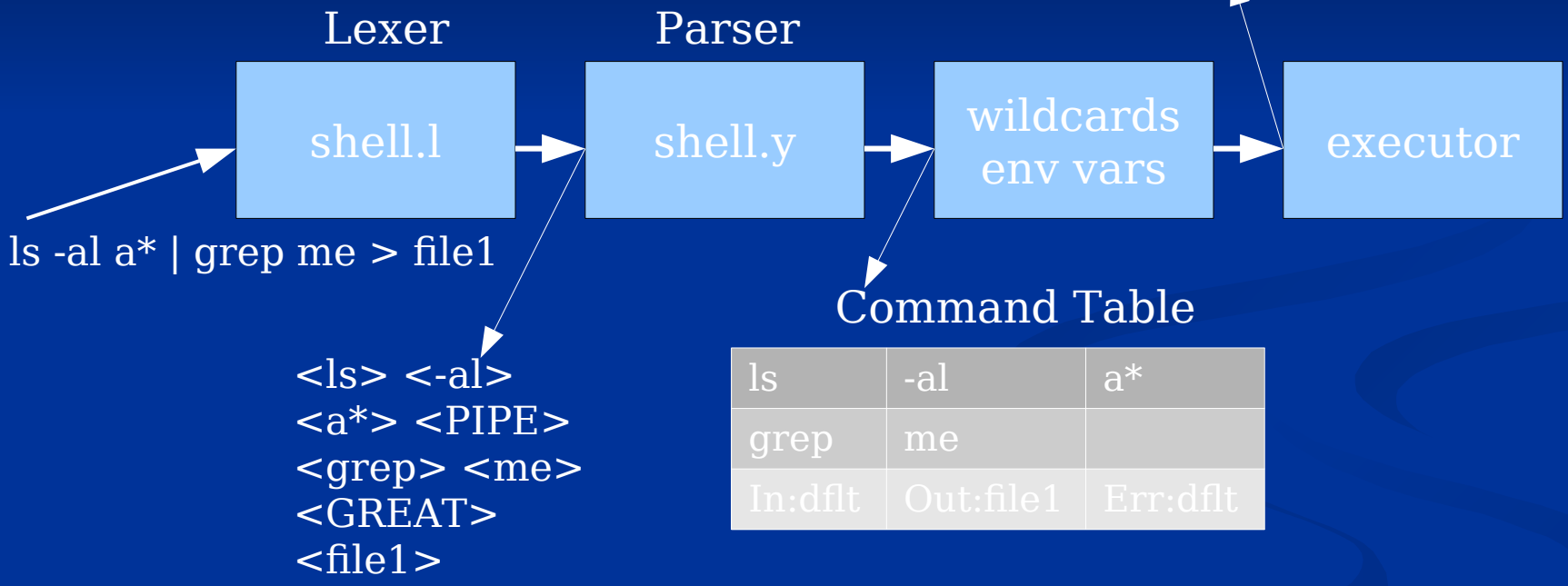
- Redirect pin[0] to child's input
- Redirect child's output to pout[1]

- Parent writes the command to pin[1]
  - And some other things – what?
- Can then read output from pout[0]
  - This is then fed back into the lexer
    - In reverse order…
  - See myunputc()

Shell

Final Command Table

ls	-al	aab	aaa
grep	me		
In:dflt	Out:file1	Err:dflt	



Wildcards

- Allow us to perform actions on more than one file at a time
- Also called file globbing
- Do not confuse with regular expressions

Standard wildcards

- $ man 7 glob
  - ?: any single character
    - DSCN00??.JPG
  - *: any number of characters, including zero
    - a*a *.c t*.log
  - [ ]: a range
    - m[a-d]m m[0-7]n
  - { }: “or” relationship – no spaces allowed
    - cp {*.doc,*.pdf} ~
  - [!]: anything not in the range
  - \: escape character

Our shell

- Only need to worry about * and ?
- Implement the simple case first
  - Add a function to shell.y to do the expansion

shell.y

- Before…

```
argument: WORD {
  insert_single_command(current_command, $1);
} ;
```

- After…

```
argument: WORD {
  expand_wildcards($1);
} ;
```

POSIX Regular expressions

- int regcomp(regex_t *preg, const char *regex, int cflags)
  - Compiles the regular expression into a form that regexec() can use
  - Completes the passed regex_t
    - Finite state automata
- int regexec(const regex_t *preg, const char *string, size_t nmatch, regmatch_t pmatch[], int eflags)
  - Do the actual comparison

```
typedef struct {
        regoff_t rm_so;
        regoff_t rm_eo;
} regmatch_t;
```

void regfree(regex_t *preg)

- regex_t's contain dynamically allocated memory
- You have to free it
  - ...or you'll have a memory leak
- Even have to free before reusing!

- Does not free the regex_t itself

Questions?