```
6600: // Shell.
6601:
6602: #include "types.h"
6603: #include "user.h"
6604: #include "fcntl.h"
6605:
6606: // Parsed command representation
6607: #define EXEC  1
6608: #define REDIR 2
6609: #define PIPE  3
6610: #define LIST  4
6611: #define BACK  5
6612:
6613: #define MAXARGS 10
6614:
6615: struct cmd {
6616:   int type;
6617: };
6618:
6619: struct execcmd {
6620:   int type;
6621:   char *argv[MAXARGS];
6622:   char *eargv[MAXARGS];
6623: };
6624:
6625: struct redircmd {
6626:   int type;
6627:   struct cmd *cmd;
6628:   char *file;
6629:   char *efile;
6630:   int mode;
6631:   int fd;
6632: };
6633:
6634: struct pipecmd {
6635:   int type;
6636:   struct cmd *left;
6637:   struct cmd *right;
6638: };
6639:
6640: struct listcmd {
6641:   int type;
6642:   struct cmd *left;
6643:   struct cmd *right;
6644: };
6645:
6646: struct backcmd {
6647:   int type;
6648:   struct cmd *cmd;
6649: };
6650:
6651: int fork1(void);  // Fork but panics on failure.
6652: void panic(char*);
6653: struct cmd *parsecmd(char*);
6654:
6655: // Execute cmd.  Never returns.
6656: void
6657: runcmd(struct cmd *cmd)
6658: {
6659:   int p[2];
6660:   struct backcmd *bcmd;
6661:   struct execcmd *ecmd;
6662:   struct listcmd *lcmd;
6663:   struct pipecmd *pcmd;
6664:   struct redircmd *rcmd;
6665:
6666:   if(cmd == 0)
6667:     exit();
6668:
6669:   switch(cmd->type){
6670:   default:
6671:     panic("runcmd");
6672:
6673:   case EXEC:
6674:     ecmd = (struct execcmd*)cmd;
6675:     if(ecmd->argv[0] == 0)
6676:       exit();
6677:     exec(ecmd->argv[0], ecmd->argv);
6678:     printf(2, "exec %s failed\n", ecmd->argv[0]);
6679:     break;
6680:
6681:   case REDIR:
6682:     rcmd = (struct redircmd*)cmd;
6683:     close(rcmd->fd);
6684:     if(open(rcmd->file, rcmd->mode) < 0){
6685:       printf(2, "open %s failed\n", rcmd->file);
6686:       exit();
6687:     }
6688:     runcmd(rcmd->cmd);
6689:     break;
6690:
6691:   case LIST:
6692:     lcmd = (struct listcmd*)cmd;
6693:     if(fork1() == 0)
6694:       runcmd(lcmd->left);
6695:     wait();
6696:     runcmd(lcmd->right);
6697:     break;
6698:
6699:   case PIPE:
6700:     pcmd = (struct pipecmd*)cmd;
6701:     if(pipe(p) < 0)
6702:       panic("pipe");
6703:     if(fork1() == 0){
6704:       close(1);
6705:       dup(p[1]);
6706:       close(p[0]);
6707:       close(p[1]);
6708:       runcmd(pcmd->left);
6709:     }
6710:     if(fork1() == 0){
6711:       close(0);
6712:       dup(p[0]);
6713:       close(p[0]);
6714:       close(p[1]);
6715:       runcmd(pcmd->right);
6716:     }
6717:     close(p[0]);
6718:     close(p[1]);
6719:     wait();
6720:     wait();
6721:     break;
6722:
6723:   case BACK:
6724:     bcmd = (struct backcmd*)cmd;
6725:     if(fork1() == 0)
6726:       runcmd(bcmd->cmd);
6727:     break;
6728:   }
6729:   exit();
6730: }
6731:
6732: int
6733: getcmd(char *buf, int nbuf)
```

```
6734: {
6735:   printf(2, "$ ");
6736:   memset(buf, 0, nbuf);
6737:   gets(buf, nbuf);
6738:   if(buf[0] == 0) // EOF
6739:     return -1;
6740:   return 0;
6741: }
6742:
6743: int
6744: main(void)
6745: {
6746:   static char buf[100];
6747:   int fd;
6748:
6749:   // Assumes three file descriptors open.
6750:   while((fd = open("console", O_RDWR)) >= 0){
6751:     if(fd >= 3){
6752:       close(fd);
6753:       break;
6754:     }
6755:   }
6756:
6757:   // Read and run input commands.
6758:   while(getcmd(buf, sizeof(buf)) >= 0){
6759:     if(buf[0] == 'c' && buf[1] == 'd' && buf[2] == ' '){
6760:       // Clumsy but will have to do for now.
6761:       // Chdir has no effect on the parent if run in the child.
6762:       buf[strlen(buf)-1] = 0;  // chop \n
6763:       if(chdir(buf+3) < 0)
6764:         printf(2, "cannot cd %s\n", buf+3);
6765:       continue;
6766:     }
6767:     if(fork1() == 0)
6768:       runcmd(parsecmd(buf));
6769:     wait();
6770:   }
6771:   exit();
6772: }
6773:
6774: void
6775: panic(char *s)
6776: {
6777:   printf(2, "%s\n", s);
6778:   exit();
6779: }
6780:
6781: int
6782: fork1(void)
6783: {
6784:   int pid;
6785:
6786:   pid = fork();
6787:   if(pid == -1)
6788:     panic("fork");
6789:   return pid;
6790: }
6791:
6792: // Constructors
6793:
6794: struct cmd*
6795: execcmd(void)
6796: {
6797:   struct execcmd *cmd;
6798:
6799:   cmd = malloc(sizeof(*cmd));
6800:   memset(cmd, 0, sizeof(*cmd));
6801:   cmd->type = EXEC;
6802:   return (struct cmd*)cmd;
6803: }
6804:
6805: struct cmd*
6806: redircmd(struct cmd *subcmd, char *file, char *efile, int mode, int fd)
6807: {
6808:   struct redircmd *cmd;
6809:
6810:   cmd = malloc(sizeof(*cmd));
6811:   memset(cmd, 0, sizeof(*cmd));
6812:   cmd->type = REDIR;
6813:   cmd->cmd = subcmd;
6814:   cmd->file = file;
6815:   cmd->efile = efile;
6816:   cmd->mode = mode;
6817:   cmd->fd = fd;
6818:   return (struct cmd*)cmd;
6819: }
6820:
6821: struct cmd*
6822: pipecmd(struct cmd *left, struct cmd *right)
6823: {
6824:   struct pipecmd *cmd;
6825:
6826:   cmd = malloc(sizeof(*cmd));
6827:   memset(cmd, 0, sizeof(*cmd));
6828:   cmd->type = PIPE;
6829:   cmd->left = left;
6830:   cmd->right = right;
6831:   return (struct cmd*)cmd;
6832: }
6833:
6834: struct cmd*
6835: listcmd(struct cmd *left, struct cmd *right)
6836: {
6837:   struct listcmd *cmd;
6838:
6839:   cmd = malloc(sizeof(*cmd));
6840:   memset(cmd, 0, sizeof(*cmd));
6841:   cmd->type = LIST;
6842:   cmd->left = left;
6843:   cmd->right = right;
6844:   return (struct cmd*)cmd;
6845: }
6846:
6847: struct cmd*
6848: backcmd(struct cmd *subcmd)
6849: {
6850:   struct backcmd *cmd;
6851:
6852:   cmd = malloc(sizeof(*cmd));
6853:   memset(cmd, 0, sizeof(*cmd));
6854:   cmd->type = BACK;
6855:   cmd->cmd = subcmd;
6856:   return (struct cmd*)cmd;
6857: }
6858: // Parsing
6859:
6860: char whitespace[] = " \t\r\n\v";
6861: char symbols[] = "<|>&;()";
6862:
6863: int
6864: gettoken(char **ps, char *es, char **q, char **eq)
6865: {
6866:   char *s;
6867:   int ret;
```

```
6868:
6869:   s = *ps;
6870:   while(s < es && strchr(whitespace, *s))
6871:     s++;
6872:   if(q)
6873:     *q = s;
6874:   ret = *s;
6875:   switch(*s){
6876:   case 0:
6877:     break;
6878:   case '|':
6879:   case '(':
6880:   case ')':
6881:   case ';':
6882:   case '&':
6883:   case '<':
6884:     s++;
6885:     break;
6886:   case '>':
6887:     s++;
6888:     if(*s == '>'){
6889:       ret = '+';
6890:       s++;
6891:     }
6892:     break;
6893:   default:
6894:     ret = 'a';
6895:     while(s < es && !strchr(whitespace, *s) && !strchr(symbols, *s))
6896:       s++;
6897:     break;
6898:   }
6899:   if(eq)
6900:     *eq = s;
6901:
6902:   while(s < es && strchr(whitespace, *s))
6903:     s++;
6904:   *ps = s;
6905:   return ret;
6906: }
6907:
6908: int
6909: peek(char **ps, char *es, char *toks)
6910: {
6911:   char *s;
6912:
6913:   s = *ps;
6914:   while(s < es && strchr(whitespace, *s))
6915:     s++;
6916:   *ps = s;
6917:   return *s && strchr(toks, *s);
6918: }
6919:
6920: struct cmd *parseline(char**, char*);
6921: struct cmd *parsepipe(char**, char*);
6922: struct cmd *parseexec(char**, char*);
6923: struct cmd *nulterminate(struct cmd*);
6924:
6925: struct cmd*
6926: parsecmd(char *s)
6927: {
6928:   char *es;
6929:   struct cmd *cmd;
6930:
6931:   es = s + strlen(s);
6932:   cmd = parseline(&s, es);
6933:   peek(&s, es, "");
6934:   if(s != es){
6935:     printf(2, "leftovers: %s\n", s);
6936:     panic("syntax");
6937:   }
6938:   nulterminate(cmd);
6939:   return cmd;
6940: }
6941:
6942: struct cmd*
6943: parseline(char **ps, char *es)
6944: {
6945:   struct cmd *cmd;
6946:
6947:   cmd = parsepipe(ps, es);
6948:   while(peek(ps, es, "&")){
6949:     gettoken(ps, es, 0, 0);
6950:     cmd = backcmd(cmd);
6951:   }
6952:   if(peek(ps, es, ";")){
6953:     gettoken(ps, es, 0, 0);
6954:     cmd = listcmd(cmd, parseline(ps, es));
6955:   }
6956:   return cmd;
6957: }
6958:
6959: struct cmd*
6960: parsepipe(char **ps, char *es)
6961: {
6962:   struct cmd *cmd;
6963:
6964:   cmd = parseexec(ps, es);
6965:   if(peek(ps, es, "|")){
6966:     gettoken(ps, es, 0, 0);
6967:     cmd = pipecmd(cmd, parsepipe(ps, es));
6968:   }
6969:   return cmd;
6970: }
6971:
6972: struct cmd*
6973: parseredirs(struct cmd *cmd, char **ps, char *es)
6974: {
6975:   int tok;
6976:   char *q, *eq;
6977:
6978:   while(peek(ps, es, "<>")){
6979:     tok = gettoken(ps, es, 0, 0);
6980:     if(gettoken(ps, es, &q, &eq) != 'a')
6981:       panic("missing file for redirection");
6982:     switch(tok){
6983:     case '<':
6984:       cmd = redircmd(cmd, q, eq, O_RDONLY, 0);
6985:       break;
6986:     case '>':
6987:       cmd = redircmd(cmd, q, eq, O_WRONLY|O_CREATE, 1);
6988:       break;
6989:     case '+':  // >>
6990:       cmd = redircmd(cmd, q, eq, O_WRONLY|O_CREATE, 1);
6991:       break;
6992:     }
6993:   }
6994:   return cmd;
6995: }
6996:
6997: struct cmd*
6998: parseblock(char **ps, char *es)
6999: {
7000:   struct cmd *cmd;
7001:
```

```
7002:   if(!peek(ps, es, "("))
7003:     panic("parseblock");
7004:   gettoken(ps, es, 0, 0);
7005:   cmd = parseline(ps, es);
7006:   if(!peek(ps, es, ")"))
7007:     panic("syntax - missing )");
7008:   gettoken(ps, es, 0, 0);
7009:   cmd = parseredirs(cmd, ps, es);
7010:   return cmd;
7011: }
7012:
7013: struct cmd*
7014: parseexec(char **ps, char *es)
7015: {
7016:   char *q, *eq;
7017:   int tok, argc;
7018:   struct execcmd *cmd;
7019:   struct cmd *ret;
7020:
7021:   if(peek(ps, es, "("))
7022:     return parseblock(ps, es);
7023:
7024:   ret = execcmd();
7025:   cmd = (struct execcmd*)ret;
7026:
7027:   argc = 0;
7028:   ret = parseredirs(ret, ps, es);
7029:   while(!peek(ps, es, "|)&;")){
7030:     if((tok=gettoken(ps, es, &q, &eq)) == 0)
7031:       break;
7032:     if(tok != 'a')
7033:       panic("syntax");
7034:     cmd->argv[argc] = q;
7035:     cmd->eargv[argc] = eq;
7036:     argc++;
7037:     if(argc >= MAXARGS)
7038:       panic("too many args");
7039:     ret = parseredirs(ret, ps, es);
7040:   }
7041:   cmd->argv[argc] = 0;
7042:   cmd->eargv[argc] = 0;
7043:   return ret;
7044: }
7045:
7046: // NUL-terminate all the counted strings.
7047: struct cmd*
7048: nulterminate(struct cmd *cmd)
7049: {
7050:   int i;
7051:   struct backcmd *bcmd;
7052:   struct execcmd *ecmd;
7053:   struct listcmd *lcmd;
7054:   struct pipecmd *pcmd;
7055:   struct redircmd *rcmd;
7056:
7057:   if(cmd == 0)
7058:     return 0;
7059:
7060:   switch(cmd->type){
7061:   case EXEC:
7062:     ecmd = (struct execcmd*)cmd;
7063:     for(i=0; ecmd->argv[i]; i++)
7064:       *ecmd->eargv[i] = 0;
7065:     break;
7066:
7067:   case REDIR:
7068:     rcmd = (struct redircmd*)cmd;
7069:     nulterminate(rcmd->cmd);
7070:     *rcmd->efile = 0;
7071:     break;
7072:
7073:   case PIPE:
7074:     pcmd = (struct pipecmd*)cmd;
7075:     nulterminate(pcmd->left);
7076:     nulterminate(pcmd->right);
7077:     break;
7078:
7079:   case LIST:
7080:     lcmd = (struct listcmd*)cmd;
7081:     nulterminate(lcmd->left);
7082:     nulterminate(lcmd->right);
7083:     break;
7084:
7085:   case BACK:
7086:     bcmd = (struct backcmd*)cmd;
7087:     nulterminate(bcmd->cmd);
7088:     break;
7089:   }
7090:   return cmd;
7091: }
```