オペレーティングシステムの機能を使ってみよう
第１０章 UNIX シェル

UNIX のシェルとは

- CLI（Command Line Interface）方式のコマンドインタプリタ
- macOS の Finder や Windows の Explorer は GUI 方式のシェル
- CLI 版のシェルは，sh，bash，ksh，zsh，csh，tcsh など
- UNIX シェルの持つべき機能
  1. 外部コマンド（プログラム）の起動
  2. カレントディレクトリの変更
  3. 環境変数の管理
  4. 入出力のリダイレクト，パイプ（<，>，|）
  5. ジョブの管理（jobs，fg，bg など）
  6. ファイル名の展開（ワイルドカード（*，?））
  7. 繰り返しや条件判断
  8. スクリプトの実行（処理の自動化）

簡易 UNIX シェル（myshell）

- 特徴
  C 言語で 70 行以内で記述できる簡易シェル
- できること
  1. 外部コマンド（プログラム）の起動
  2. カレントディレクトリの変更
- 目的
  myshell の構造を学び，「fork-exec 方式」，「環境変数」，「リダイレクト」等への理解を深める．
- 目標
  「環境変数の管理機能」，「リダイレクト機能」を myshell に追加できるようになる．

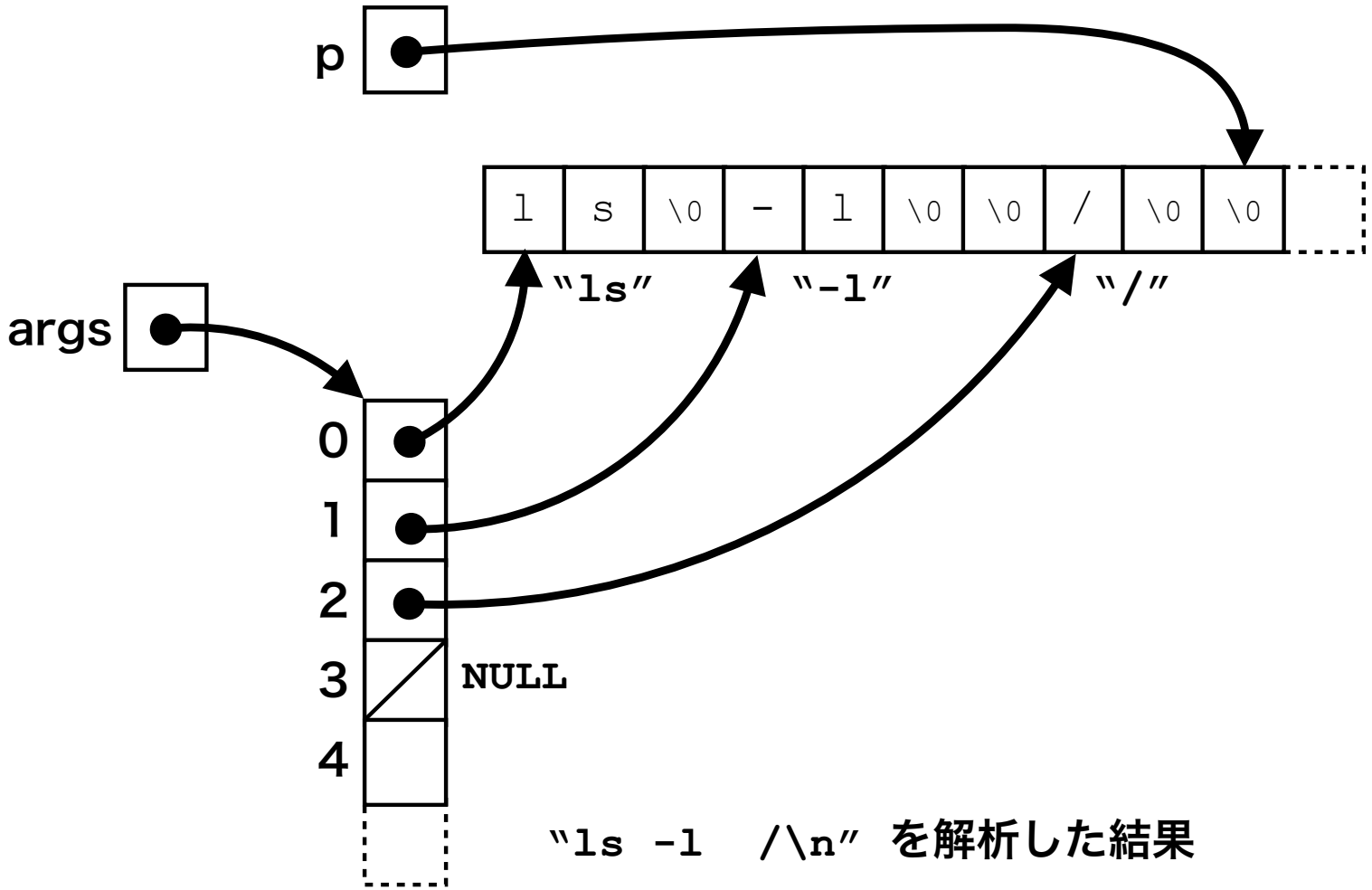
基本構造（main() 関数）

```
1  int main() {
2    char buf[MAXLINE+2];                          // コマンド行を格納する配列
3    char *args[MAXARGS+1];                        // 解析結果を格納する文字列配列
4    for (;;) {
5      printf("Command: ");                        // プロンプトを表示する
6      if (fgets(buf,MAXLINE+2,stdin)==NULL) {     // コマンド行を入力する
7        printf("\n");                             // EOF なら
8        break;                                    //   正常終了する
9      }
10     if (strchr(buf, '\n')==NULL) {              // '\n'がバッファにない場合は
11       fprintf(stderr, "行が長すぎる\n");        //   コマンド行が長すぎたので
12       return 1;                                 //   異常終了する
13     }
14     if (!parse(buf,args)) {                     // コマンド行を解析する
15       fprintf(stderr, "引数が多すぎる\n");      //   文字列が多すぎる場合は
16       continue;                                 //   ループの先頭に戻る
17     }
18     if (args[0]!=NULL) execute(args);           //   コマンドを実行する
19   }
20   return 0;
21 }
```

コマンド行の解析（parse() 関数）（1）

```
1  int parse(char *p, char *args[]) {                  // コマンド行を解析する
2    int i=0;                                          // 解析後文字列の数
3    for (;;) {
4      while (isspace(*p)) *p++ = '\0';                // 空白を'\0'に書換える
5      if (*p=='\0' || i>=MAXARGS) break;              // コマンド行の終端に到達で終了
6      args[i++] = p;                                  // 文字列を文字列配列に記録
7      while (*p!='\0' && !isspace(*p)) p++;           // 文字列の最後まで進む
8    }
9    args[i] = NULL;                                   // 文字列配列の終端マーク
10   return *p=='\0';                                  // 解析完了なら 1 を返す
11 }
```

コマンド行の解析（parse() 関数）（2）



コマンドの実行（execute() 関数）

```
1  void execute(char *args[]) {                    // コマンドを実行する
2    if (strcmp(args[0], "cd")==0) {               // cd（内部コマンド）
3      if (args[1]==NULL || args[2]!=NULL) {       //   引数を確認して
4        fprintf(stderr,"Usage: cd DIR\n");        //     過不足ありなら使い方表示
5      } else if (chdir(args[1])<0) {              //   親プロセスが chdir する
6        perror(args[1]);                          //     chdirに失敗したらperror
7      }
8    } else {                                      // 外部コマンドなら
9      int pid, status;
10     if ((pid = fork()) < 0) {                   //   新しいプロセスを作る
11       perror("fork");
12       exit(1);
13     }
14     if (pid==0) {                               //   子プロセスなら
15       execvp(args[0], args);                    //     コマンドを実行
16       perror(args[0]);
17       exit(1);
18     } else {                                    //   親プロセスなら
19       while (wait(&status) != pid)              //     子の終了を待つ
20         ;
21     }
22   }
23 }
```