５ＭＳ実験ＯＳ　第３回　Ｃ言語開発環境，システムコール

１．　Ｃ言語復習，コンパイラ

```
#include <stdio.h>
int main()
{
    printf("Hello World!!¥n");

    return 0;
}
```

1行目

ヘッダファイルと呼ぶファイルを取り込むための指示を行います。詳しい説明は後ほど行いますので、ここでは標準入出力を行うときに必要な手続きであると思っていただく程度で十分です。

2行目

プログラムの入口です。複数のソースファイルで構成するプログラムも、ここから実行を開始します。この行は必須です。

3行目から7行目

プログラムの本体です。本体は3行目の{（左中括弧）と、7行目の}（右中括弧）で括ります。

4行目

標準出力に「Hello World!!」メッセージを出力する命令です。出力するメッセージは"（引用符）で括ります。このようにプログラムの本体に記述する命令を「文」と呼んでいます。文は;（セミコロン）で区切ります。

6行目

プログラムの出口です。指定した値（この場合は0）を呼び出し元に戻して終了します。

＊コンパイルと実行

Ｃ言語で記述したソースファイルを元に、コンピュータで実際に動作するプログラムを作成する。

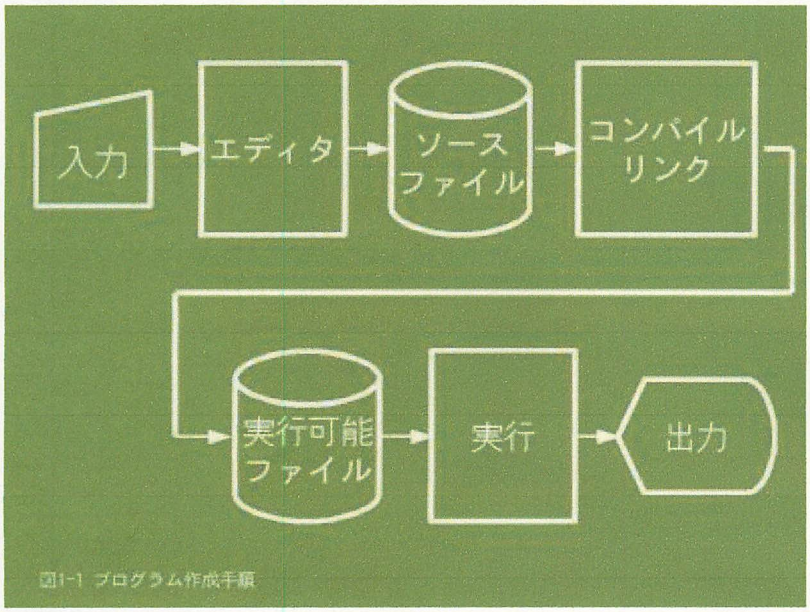


図1-1 プログラム作成手順

cprog というディレクトリを作成し，そこで以下のレポートの作業を行う。

レポート 1-1：

１．gedit でソースファイル hello.c（最初にあるプログラム）を作る。

２．gcc hello.c でコンパイル，リンクする。

３．実行ファイル a.out ができる。

４．./a.out で実行する。

実行結果を書け。

２．　プリプロセッサとヘッダファイル

「プリプロセッサ(preprocessor)」という言葉は、直訳するならば「前処理をするもの」。コンパイラはプログラムをまず最初にプリプロセッサにかけて加工し、その結果に対して本来のコンパイル処理(機械語への変換)を実行するのです。

レポート 2-1：gcc -E hello.c での出力の中で extern int printf の記述のある行を書け。

#include <ファイル名>は，<ファイル名>で示されたファイルを取り込みます。

レポート 2-1：sudo find コマンドで stdio.h（ヘッダファイル）があるディレクトリを調べ，すべて書け。また，/usr/include ディレクトリにある std から始まる文字列のヘッダファイルを書け。

#define <文字列 1> <文字列 2>は，識別子<文字列 1>を値<文字列 2>と定義します

レポート 2-3：以下のプログラムを記述したファイルを gcc -E ファイル名　の応答の最後 men= の行を書け。

```
#include<stdio.h>
#define PI 3.14
#define EN PI*kei*kei
int main(){
  float kei,men;
  kei = 2;
  men = EN;
  printf("men = %f¥n",men);
  return 0;
}
```

3.　makefile

大規模なプログラムを作成していると、ソースファイルを少し書き換えただけですべてのファイルをコンパイルし直すなのは面倒！！コンパイル作業を自動化するための仕組みが makefile。

レポート 3-1：hello.c のあるディレクトリに，以下を記述した Makefile という名前のファイルを作成する。その後 make とコマンドを打つ。ls コマンドで make により作成されたファイル確認しファイル名を書け。./hello と打つ。実行結果を書け。（注意：gcc の前は Tab キー）

```
hello: hello.c
	gcc -o hello hello.c
```

記述内容の意味は

ターゲット名: 依存ファイル名

　　コマンド行

hello2.c	sayhi.c

#include<stdio.h>
int main(int argc,char *argv[]){
　printf("Hello,World!!¥n");
　sayhi();
　return 0;
}
	#include<stdio.h>
	void sayhi(){
	　printf("hi!!¥n");
	}

Makefile

```
hello2: hello2.o sayhi.o
	gcc -o hello2 hello2.o sayhi.o
hello2.o: hello2.c
	gcc -c hello2.c
sayhi.o: sayhi.c
	gcc -c sayhi.c
```

レポート 3-2：cprog2 ディレクトリを作り，そこに上記の hello2.c　sayhi.c　Makefile　の 3 つのフ

ファイルを作成し，make して(1)make コマンドの返答を書け。(2)作成された実行ファイル hollo2 を実行した結果を書け。(3)続けて，make して make コマンドの返答を書け。(4)続けて，gedit で sayhi.c の出力文字を変更し保存した後，make し，make コマンドの返答を書け。

＊　マクロ（変数）と内部マクロ，関数

```
# Makefile
#マクロの定義 CC，CFLAGS はコンパイラ，オプションに使われる
program = hello
objs = hello.o sayhi.o
CC = gcc
CFLAGS = -g -Wall

$(program): $(objs)
	$(CC) -o $(program) $^

hello.o: hello.c
	$(CC) $(CFLAGS) -c $<

edajima.o: sayhi.c
	$(CC) $(CFLAGS) -c $<
```

内部マクロの例

$@：ターゲットファイル名

$%：ターゲットがアーカイブメンバだったときのターゲットメンバ名

$<：最初の依存するファイルの名前

$?：ターゲットより新しいすべての依存するファイル名

$^：すべての依存するファイルの名前

$+：Makefile と同じ順番の依存するファイルの名前

$*：サフィックスを除いたターゲットの名前

文字列処理などの関数が使える，以下一例を示す。

subst：置換動作

patsubst：置換動作，ワイルドカードあり

strip：空白文字の削除

findstring：文字列を探す

dir：ディレクトリ部分の抽出

nodir：ファイル部分の抽出

suffix：サフィックス（拡張子）部分

basename：サフィックス以外

レポート 3-3：make install，make clean とコマンドを打つことがあるが，一般的にこれらはどのような処理を行うときか，簡単に説明せよ。

レポート 3-4：Makefile に記述する ifdef の使用方法について調べよ。簡単な使用例を書き，その意味を説明

せよ。

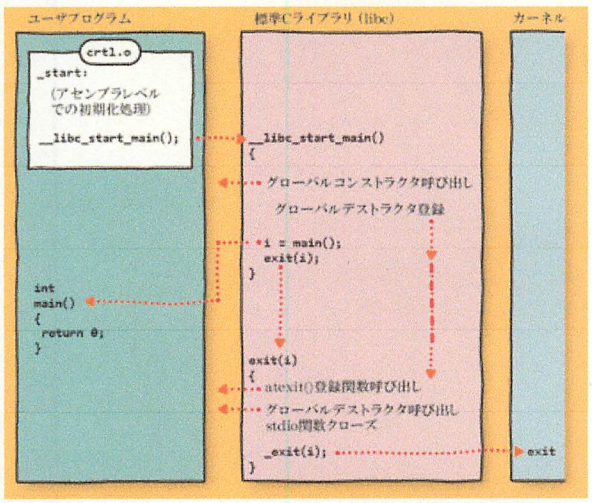
4．　システムコール

システムコール（system call）は，OS（カーネル）の持つ機能にアクセスする時に使われるしくみです。Linux などの OS 上でユーザプログラムが実行される際，さまざまなシステムコールが呼び出され，プログラムの実行が進んでいます。ユーザプログラムの観点で，OS にどのような機能があるか，OS がどのように動いているのかを知りたいと思った際，システムコールを一つ一つ見ていくことで具体的な知識を得ることができます。本記事では，そんなシステムコールのちょっと深い基本へと潜ってみましょう。

C 言語で，終了ステータス 0 を返す以外は何もしないプログラムを 1 行にまとめてみると，たとえば，

int main(){return 0;}

のように記述できます。たったこれだけで取り立てて何も起きないように見えますが，OS から見るとプログラム（プロセス）を終了するため，標準でリンクされる C 言語のスタートアップファイル（crt1.o など）や標準 C ライブラリ（libc）を経由して，exit 系のシステムコールは呼び出されます。



また，画面にテキストメッセージを出力するだけの簡単なプログラムでは，画面（標準出力等）への出力はファイルアクセスの一種であるためシステムコールが必要で，この場合は write のシステムコールが使用されます。

＊　プロセス生成,fork(),wait()

fork 関数は、呼び出し元プロセスを複製して、子プロセス（新しいプロセス）を生成します。fork 関数を呼び出した（実行した）プロセスを親プロセス、新しく生成したプロセスを子プロセスと呼んでいます。子プロセスにはユニークなプロセス ID が付与されます。

子プロセスが親プロセスより先に終了すると、子プロセスは「ゾンビ」状態で残り続けます。これを避けるには、親プロセスは wait 関数を呼び出して子プロセスが終了したら、システムがその子プロセスに関連するリソースを解放できるようにします。

サンプルプログラム

ork で子プロセスを作成し、親プロセスは、子プロセスの終了を wait システムコールで待ちます。 wait システムコールの第 1 引数で受け取った、ステータスを表示します。

レポート 4-1：以下のプログラムの実行する。(1)①～④の実行される順番を書け。(2)子プロセスのプロセス ID はいくらか。(3) wait( )システムコールが返す値について調べよ。(4) exit( )関数について調べよ。また，EXIT_SUCCESS は stdlib.h で値を定義されているが，いくらと定義されているか調べよ。

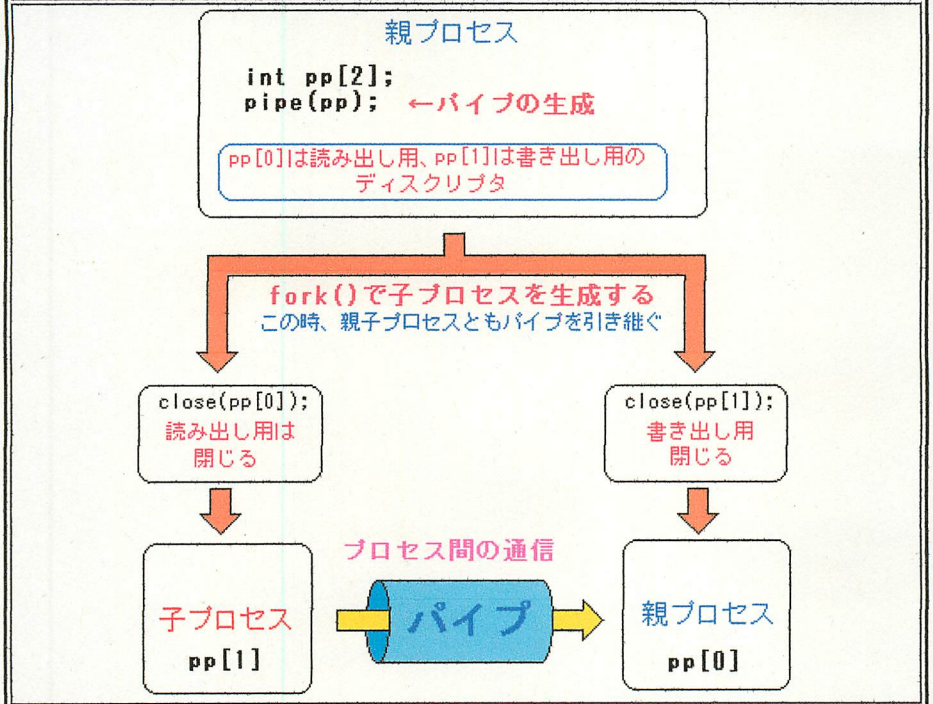
```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <stdarg.h>

#include <sys/types.h>  // fork/wait
#include <unistd.h>     // fork/sleep
#include <sys/wait.h>   // fork/wait

#include <err.h>
#include <errno.h>

int main(int argc, char *argv[])
{
        int     status = 10;
        pid_t   wait_pid;
        pid_t   pid;

        pid = fork ();//子プロセス生成

        if (-1 == pid)
        {
                err (EXIT_FAILURE, "can not fork");
        }
        else if (0 == pid)
        {
                // child
                (void) puts ("child start");//①
                sleep (5);      // 子プロセスの長い処理
                (void) puts ("child end");//②
                exit (EXIT_SUCCESS);
                /* NOTREACHED */
        }

        // parent
        (void) printf ("parents, child is %d¥n", pid); //③
        wait_pid = wait (& status);

        if (wait_pid == -1)
        {
                // wait が失敗した
                err (EXIT_FAILURE, "wait error");
        }

        (void) printf ("child = %d, status=%d¥n", wait_pid, status); //④

        exit (EXIT_SUCCESS);
}
```

＊　pipe 通信

同じマシン上での、名前なしパイプを使ったプロセス間通信。下図がプログラムの流れ。

名前なしパイプによるプロセス間通信の仕組み



(1)	親プロセスが通信を行うためのパイプを作成する。
(2)	fork()を使って、子プロセスを生成。 この時、親プロセスが作成したパイプは子プロセスにも引き継がれる。
(3)	子プロセスから親プロセスへデータの送信を行うため 子プロセス側の読み込みディスクリプタは不要なので閉じる。 親プロセス側の書き込みディスクリプタは不要なので閉じる。
(4)	そして子プロセスから親プロセスへデータの送信を行う。 書き込みの場合は write()を使い、読み込みは read() を使う。

レポート 4-2：以下のプログラムを実行した結果を書け。

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main() {
  pid_t pid;
  char buf1[256];
  char buf2[256];

  int pp[2];

  // (1)パイプの作成
  pipe(pp);

  pid = fork(); // (2)子プロセスの生成

  // 子プロセスの動き
  if (pid == 0) {
```

```
    close(pp[0]); // (3)読み込みディスクリプタを閉じる
    printf("Message To Parent : ");
    fgets(buf1, 256, stdin);//標準入力（キーボード）より読み込み
    write(pp[1], buf1, strlen(buf1) + 1); // (4)パイプへの書き込み
    close(pp[1]);
  }
  // 親プロセスの動き
  else {
    close(pp[1]); // (3)書き込みディスクリプタを閉じる
    read(pp[0], buf2, 256); // (4)パイプから読み込み
    printf("Message From Child : %s", buf2);
    close(pp[0]);
  }

  return 0;
}
```

レポート 4-3：上記プログラムを少し変更し，親から子へもデータを送信する処理を加えたプログラムを書け。

6．調査

レポート 6-1：複数のプロセスが並行して動作するときに生じるデッドロックについて調査し，Ａ４，１枚程度にまとめよ。