Nios II プログラミングその4 マルチタスク編

マルチタスクって何さ

- 並列でプログラムをできるようにさせること。

  (実際にはすごい速い速度で切り替えたりプロセッサごとに作業を割り当てさせたりしてるけどその辺の内部的なことはどうでもいい)

Nios IIで使うにはRTOSっていうOSを使えるようにしなければならない
一応Microc/OS-IIっていうRTOSが用意されている。

マルチタスク化するメリット

- シングルタスクだとひとつ遅い処理があると全部の処理が止まる
- 他の処理を途中で止めたり最初からやり直したりできる(利便性の向上)
- プログラムの保守性が上がる

マルチタスク化するデメリット

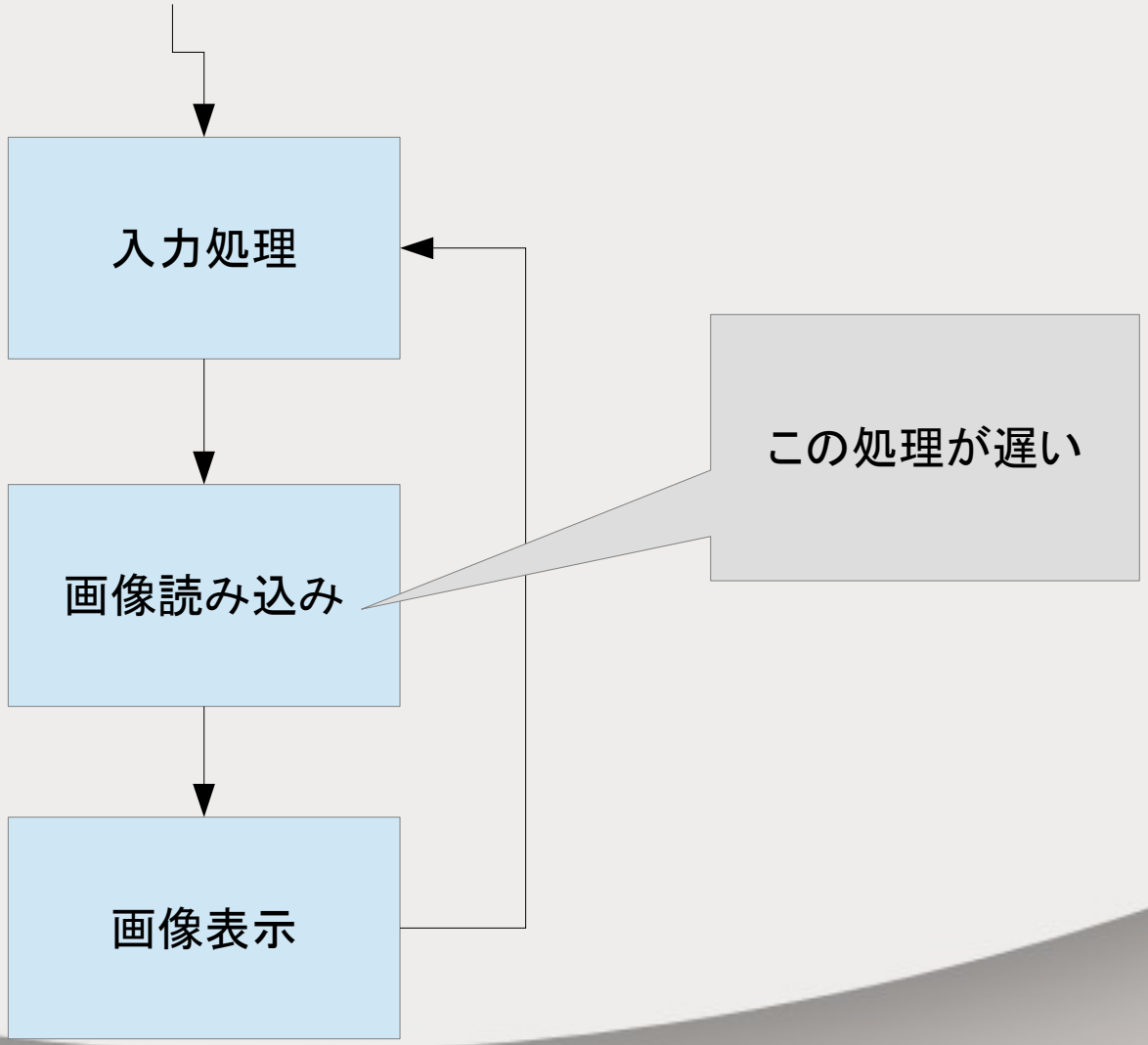
- 並列化してるから複雑すぎるとプログラミングしにくい
- タスクに優先順位を与えないといけない
- OSの処理もするのでパフォーマンスの気持ち低下する(すごく遅いマシンだと致命的)

例1

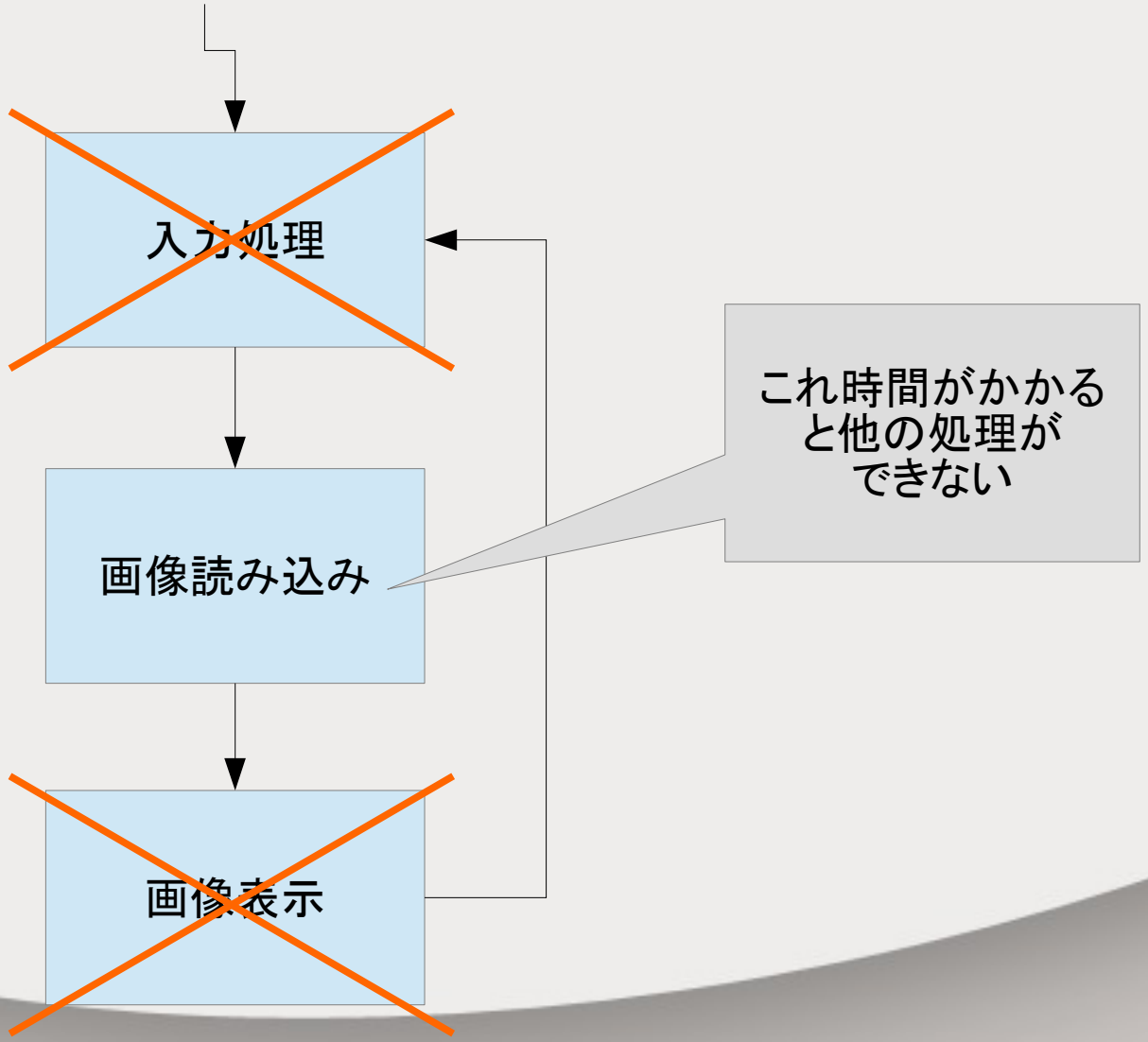
- ある大きく分けて以下の処理があったとする
  1. 入力の変化を監視する処理(ボタン入力の監視とか)
  2. 画像をSDカードから読み込む処理
  3. 画像を表示する処理

※画像読込がすごい遅いって問題があるとする

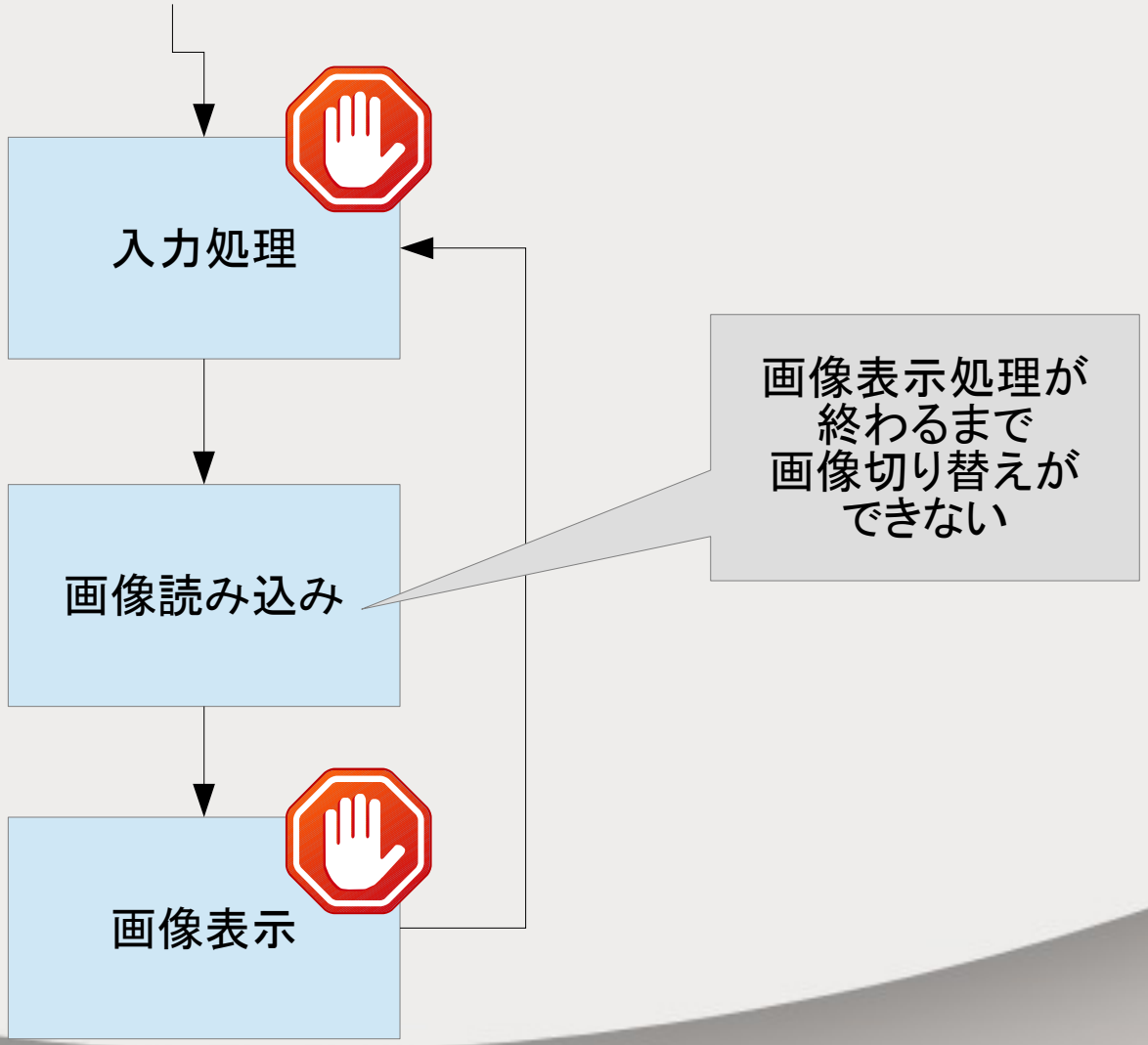
例1(シングルタスクだと)



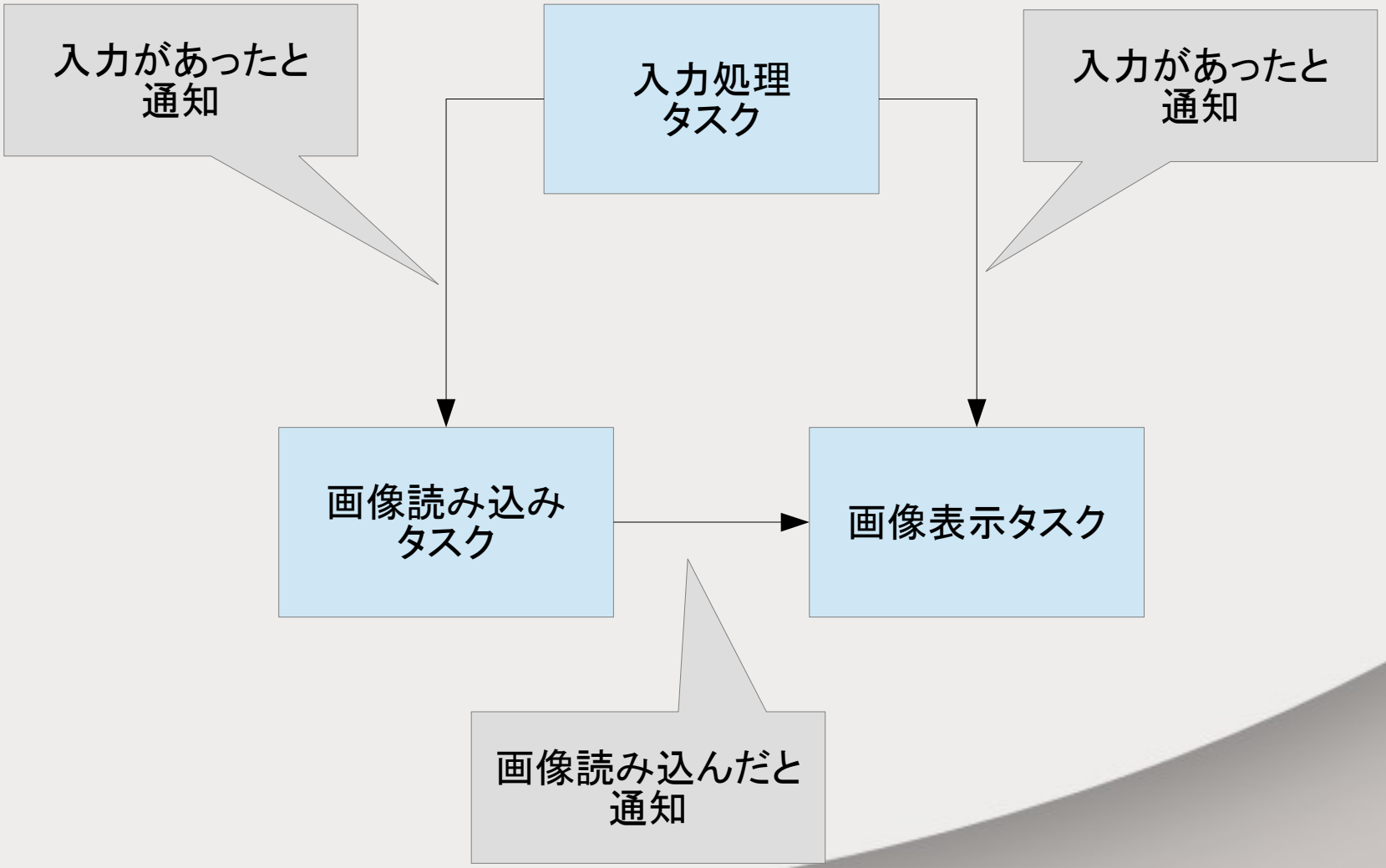
例1(シングルタスクだと)



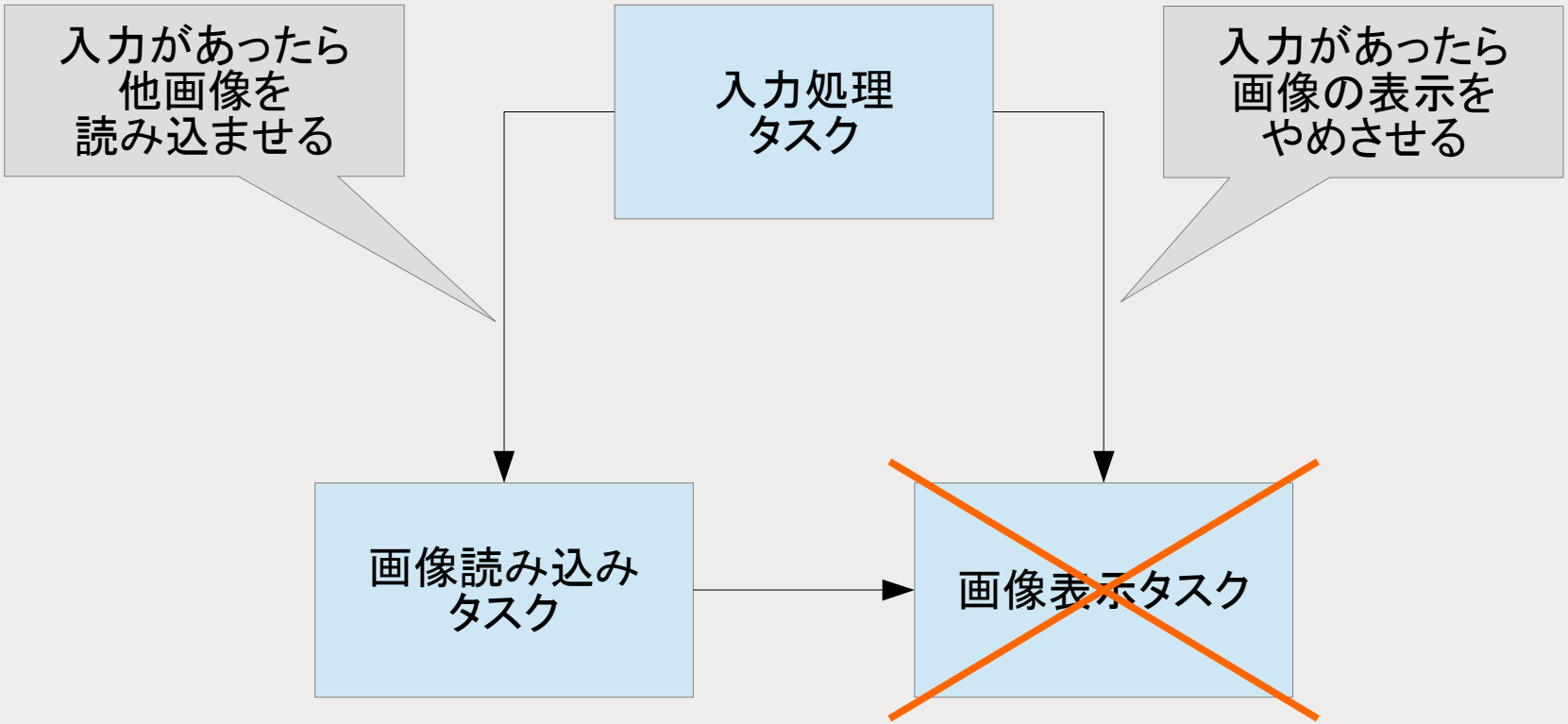
例1(シングルタスクだと)



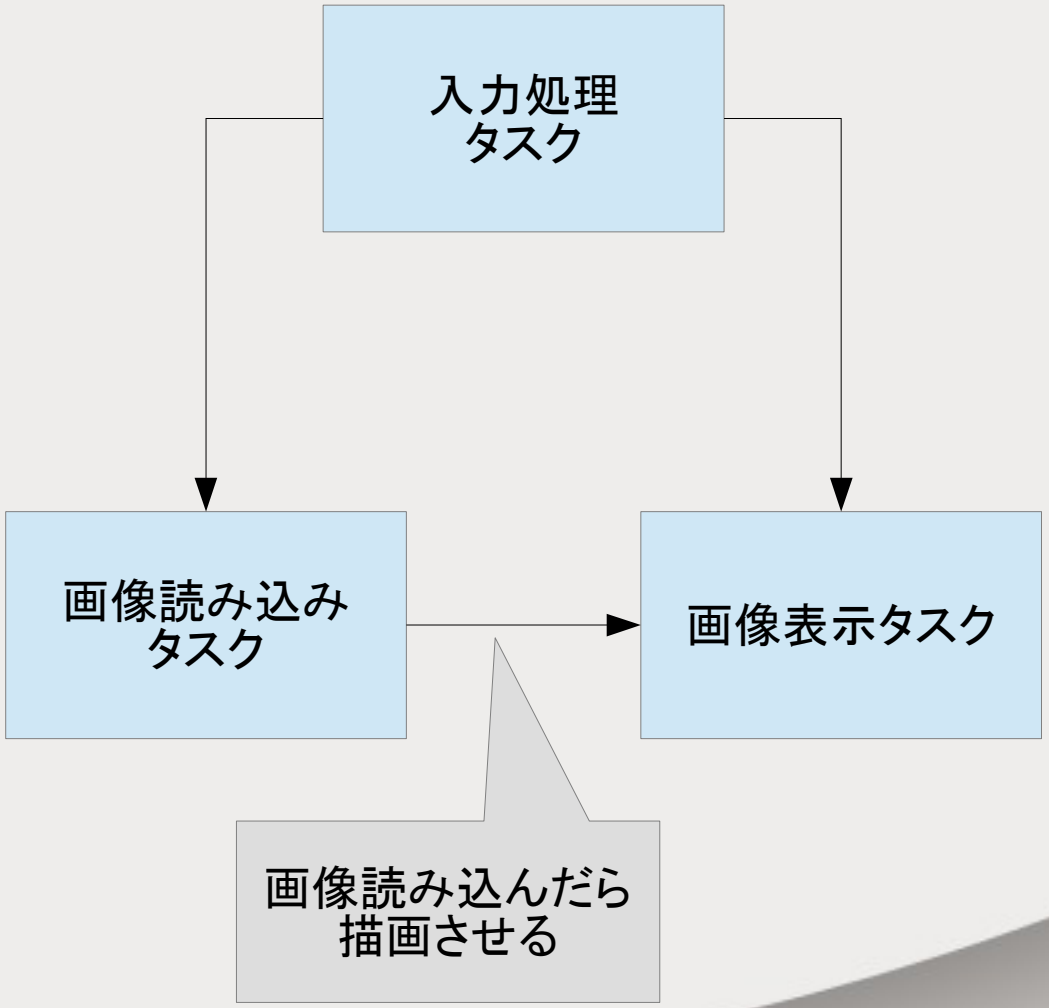
例1(マルチタスクだと)


入力があったと
通知
入力処理
タスク
入力があったと
通知
画像読み込み
タスク
画像表示タスク
画像読み込んだと
通知

例1(マルチタスクだと)


入力があったら
他画像を
読み込ませる
入力処理
タスク
入力があったら
画像の表示を
やめさせる
画像読み込み
タスク
画像表示タスク

例1(マルチタスクだと)



肝心のNios IIでマルチタスクするには

- BSPをMicroc/OS-IIのものにする
  (Nios II IDEの使い方その3参照)
- 必要なライブラリを読む

例2

- task1とtask2を実行する。

- Task1(優先順位1)
  - 一定時間ごとに文字を表示させる
  - 途中でTask2を殺す
- Task2(優先順位2)
  - 一定時間ごとに文字を表示させる

添付資料delete_task_test.cも参照

例2

どっちも1秒ごとに文字を表示してる

```
void task1(void* pdata)
{
    int i;
    for(i=0;i<10; i++)
    {
        printf("Hello from task1\n");
        OSTimeDlyHMSM(0, 0, 1, 0);
    }
    OSTaskDel(TASK2_PRIORITY);
    printf("killed task2\n");

    while (1)
    {
        printf("Hello from task1\n");
        OSTimeDlyHMSM(0, 0, 1, 0);
    }
}
```

```
void task2(void* pdata)
{
    while (1)
    {
        printf("Hello from task2\n");
        OSTimeDlyHMSM(0, 0, 1, 0);
    }
}
```

例2

Task2を削除する
(TASK2_PRIORITYはTask2の識別番号)

```
void task1(void* pdata)
{
    int i;
    for(i=0;i<10; i++)
    {
        printf("Hello from task1\n");
        OSTimeDlyHMSM(0, 0, 1, 0);
    }
    OSTaskDel(TASK2_PRIORITY);
    printf("killed task2\n");

    while (1)
    {
        printf("Hello from task1\n");
        OSTimeDlyHMSM(0, 0, 1, 0);
    }
}
```

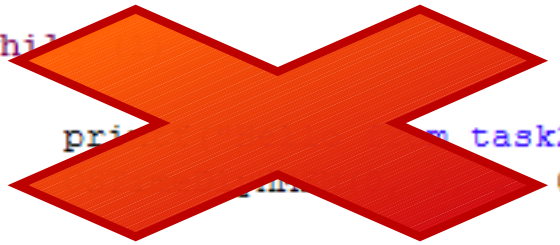
```
void task2(void* pdata)
{
      while (1)
      {
          printf("Hello from task2\n");
          OSTimeDlyHMSM(0, 0, 1, 0);
      }
}
```

例2

```
void task1(void* pdata)
{
    int i;
    for(i=0;i<10; i++)
    {
        printf("Hello from task1\n");
        OSTimeDlyHMSM(0, 0, 1, 0);
    }
    OSTaskDel(TASK2_PRIORITY);
    printf("killed task2\n");

    while (1)
    {
        printf("Hello from task1\n");
        OSTimeDlyHMSM(0, 0, 1, 0);
    }
}
```

```
void task2(void* pdata)
{
    whi
    {
        pri                    task2\n");
                                0);
    }
}
```



あとはずっとこの処理だけする

で、肝心のマルチタスクの使い方

- タスクを作るのにはOSTaskCreateExt()を使う
- タスクを削除するにはOSTaskDel()を使う
- タスクを一時停止するにはOSTaskSuspend()を使う
- タスクを再開するにはOSTaskResume()を使う
- OSStart()で開始

タスクを作るとき、優先順位が適切でないと優先順位が高いタスクにCPUが乗っ取られてそれしか処理できなくなる。
優先順位の高いタスクがCPUを100%使用してたら優先順位が低いタスクからは削除できないこともある。

で、肝心のマルチタスクの使い方

必要な引数の説明は省略します
適当に検索してください。

~~(書くののメンドクサイ)~~

余談

- タスクの優先順位が不適切な場合、優先順位の高いタスクに競合してるデバイスやCPUの処理が横取りされる。

  →優先順位を適切に設定するのと、セマフォを使って横取りせずに待ってもらう処理が必要。

- CPUのリソースを独占される例

  添付資料のbusy_multitask_test.cを実行してみると…

余談

```
1  #include <stdio.h>
2  #include "includes.h"
3
4  /* Definition of Task Stacks */
5  #define    TASK_STACKSIZE       2048
6  OS_STK     task1_stk[TASK_STACKSIZE];
7  OS_STK     task2_stk[TASK_STACKSIZE];
8
9  /* Definition of Task Priorities */
10
11 #define TASK1_PRIORITY      1
12 #define TASK2_PRIORITY      2
13
```

(↑※画像の部分は同じだけど「delete_task_test.c」とは違うコード。念のため注釈。)

この部分を入れ替えて実行してみると…
シングルプロセッサだとOSTaskDel()がうまく実行されないことがあるはず。