第4回 CPUによるプロセス実行

塩澤 秀和

OSの代表的機能

- プロセス管理
  - 実行中のプログラムの管理
  - コンピュータの実行状態の制御
- メモリ管理
  - 主記憶（メインメモリ）の管理
  - プログラム実行のためのメモリの管理
- ファイル管理
  - 補助記憶（ストレージ）の管理
  - HDDやSSDに保存されたデータの管理
- その他
  - 通信・ネットワーク，セキュリティ，ユーザ・課金管理など…

プロセス管理

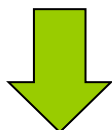
- プロセス（process）とは
  - 起動して“実行中”のプログラム
  - コンピュータの中で“動いているもの”（CPUを使っているもの）
  - OSによっては，同じ意味で「タスク」（task）ともいう

- OSによるプロセスの管理
  - プロセスの生成（プログラムの開始とメモリ確保）
  - プロセスの消滅（プログラムの停止とメモリ解放）
  - プロセスの切り替え，優先順位の管理，などなど…

- プロセスの“正体”を，よりハードウェア的に考えると…
  - 「動いている」 or 「実行中」 とは ＝ CPUで演算処理をしている
  - 「プログラムとデータがある」とは ＝ メモリの領域を占めている

コンピュータハードウェア

- 制御機能/演算機能
  - CPU(プロセッサ)
- 記憶機能
  - 主記憶(メモリ)
  - 補助記憶(HDD等)
- 入力/出力機能
  - 各種周辺機器
  - I/Oとも呼ばれる



デバイスドライバで制御

CPU

バス

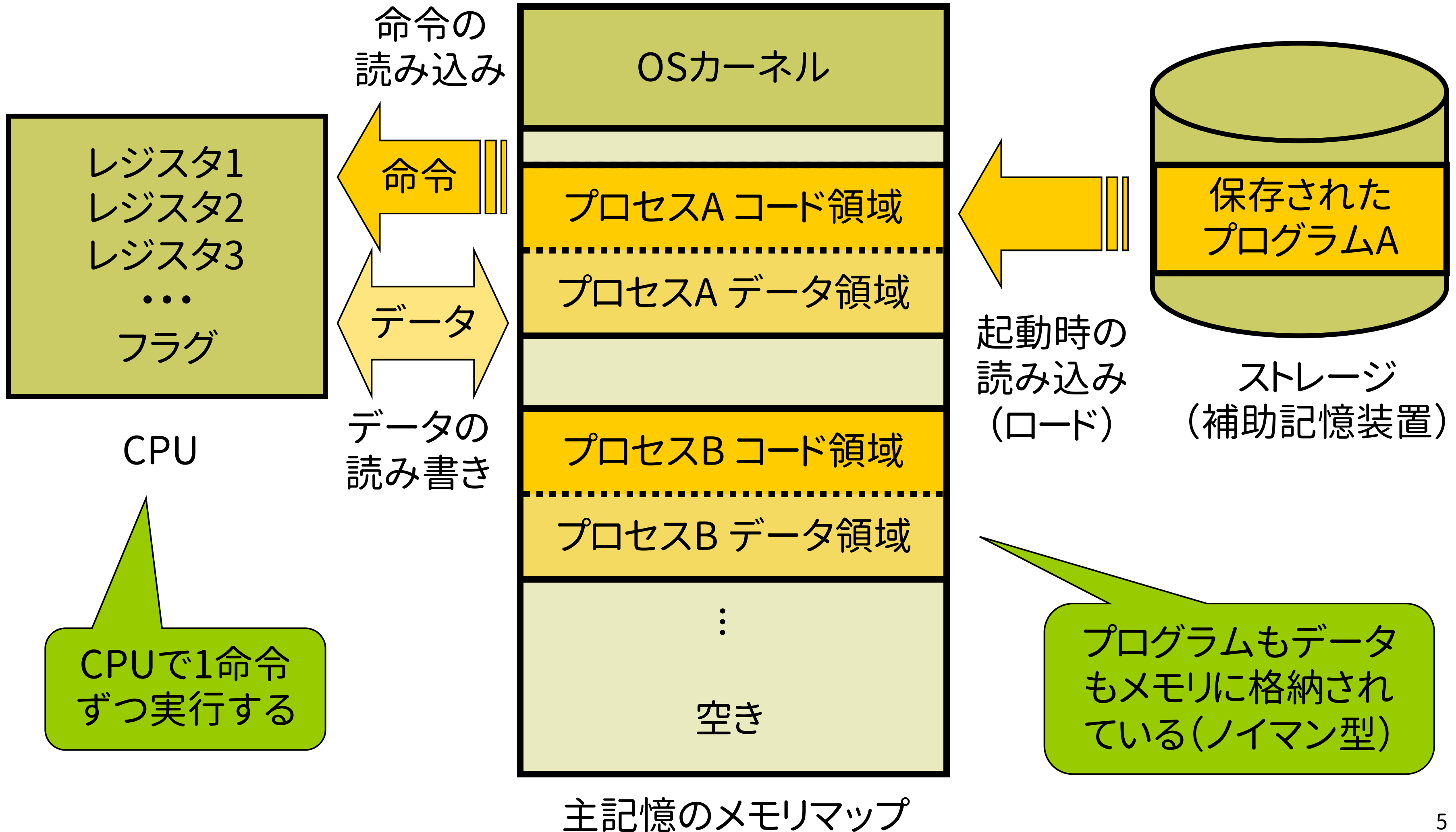
メモリ

……

入出力

ストレージ(HDD等)

プログラムのロードと実行



CPUの内部構成

- ALU（算術論理演算ユニット）
  - CPUの中心となる演算回路のかたまり
- レジスタ
  - 小容量で非常に高速な記憶領域（数個～数十個）

レジスタの種類	用途
汎用レジスタ	通常の計算やメモリのアドレス指定に使う
プログラムカウンタ	現在実行中のプログラムのアドレスを保持する
スタックポインタ	現在使用中の実行スタックのアドレスを保持する
フラグレジスタ	演算結果の正負等で変化するフラグの集まり

- その他
  - 制御用回路，入出力用のバスインタフェース，キャッシュメモリ等

CPUの動作モード

- 特権モード
  - 別名「カーネルモード」「スーパーバイザモード」
  - コンピュータの停止や外部との入出力命令など，なんでもできる状態
  - 電源ON時の起動と，カーネル実行のためのモードとして使われる

- ユーザモード
  - 通常のプロセスの実行モード
  - ハードウェア資源に直接アクセスする“危険な命令”が実行できない

- 動作モードの切り替え
  - OSカーネルは，各プロセスをユーザモードで動かす
  - プロセスは，“危険な命令”を実行したい場合はカーネルに依頼する
⇒ カーネルだけが特権モードに切り替え，危険な処理を実行できる

CPUのプログラム実行

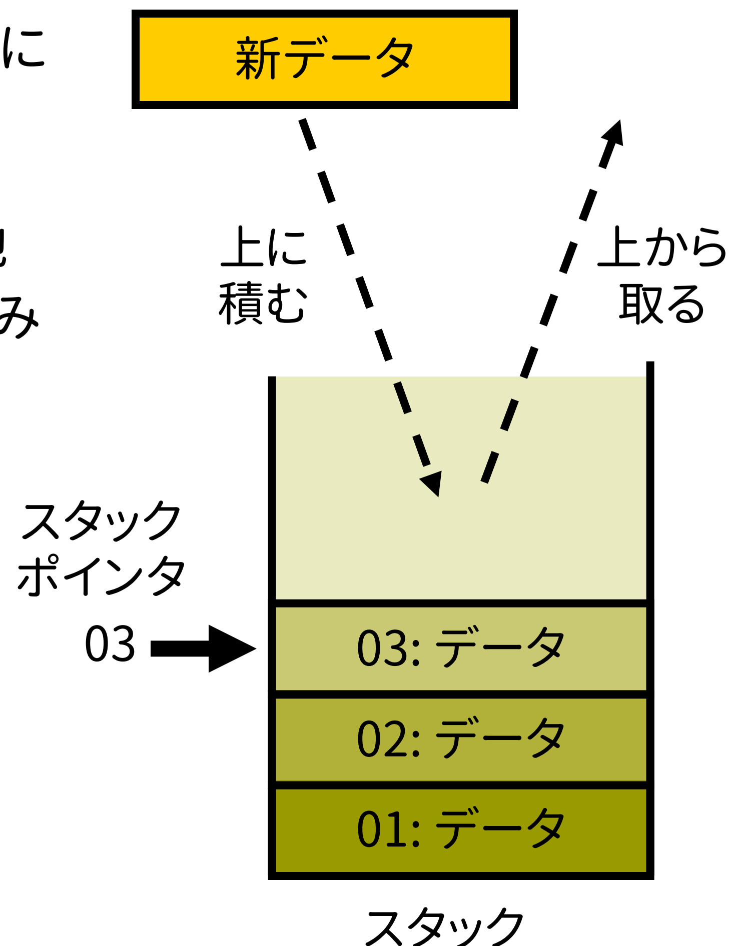
- 機械語
  - CPUが実行できるのは，「機械語」と呼ばれる2進数の命令
  - 機械語はCPU（MPU，マイクロプロセッサ）の機種ごとに異なる

- CPUの実行手順
  1. フェッチ　メモリから命令（コード番号）を1つ読みこむ
  2. デコード　命令（コード番号）を解読する
  3. 実行　メモリやレジスタの内容を読み出し，演算を実行する
  4. ライトバック　演算結果をメモリやレジスタに書き戻す

- プログラムカウンタ
  - CPUが次に実行すべき命令のアドレスを保持する特別なレジスタ
  - 命令の実行が終わると自動的に次のアドレスを指す

コールスタック（実行スタック）

- プログラム実行用のスタック
  - CPUは，命令を実行するためにメモリ上に一時的な作業領域が必要
  - 例）一時変数（C言語の普通の変数）
  - 例）関数を呼び出したときの戻り先番地
  - 「スタック」と呼ばれるデータ管理のしくみ（データ構造）でデータを保存する



- スタックのしくみ
  - データを積み上げるように保存する
  - 後入れ先出し方式（LIFO / FILO）
- スタックポインタ
  - CPUのレジスタの一種
  - 現在使用中のスタックの一番上を指す

CPUの各種フラグ

- フラグとは?
  - Flag=「旗」,Yes(1)かNo(0)かを示す1ビットの変数
  - 直前の演算結果の特徴やCPUの状態によって値が変化する
  - どんなフラグの種類があるかは,CPUによって異なる

- 代表的なフラグ
  - ゼロフラグ: 直前の演算結果がゼロか(1)ゼロでないか(0)?
  - 符号フラグ: 直前の演算結果がマイナスか(1)プラスか(0)?
  - オーバーフローフラグ: 直前の演算が桁あふれ(※)したか?
    ※ 計算結果がレジスタの保持できる値の範囲(桁数)を越える事態

- フラグレジスタ
  - CPUにある各種フラグの集合(レジスタの各ビットがフラグ)

アセンブリ言語

- アセンブリ言語（アセンブラ）とは？
  - CPUの機械語と，ほぼ1対1で対応したコンピュータ言語
  - CPUでプログラムがどう動いているのが具体的に分かる
- アセンブラの文法
  - 1行に1命令ずつ書く
  - 「オペコード　オペランド1　オペランド2…」
  - オペコード：命令の名前（演算や制御の種類）
  - オペランド：命令の引数（レジスタやアドレスなど）
- 例）CASL II
  - 情報処理技術者試験で出題されるアセンブラ（午後の選択問題）
  - 現実には存在しない仮想コンピュータCOMET II用のアセンブラ
  - 単純で基本的な機能だけを備える（教育用）

アセンブラ入門

- CASL II
  - 情報処理技術者試験で出題されるアセンブラ（午後の選択問題）
  - 現実には存在しない仮想コンピュータCOMET II用のアセンブラ
  - 単純で基本的な機能だけを備える（教育用）

- シミュレータ
  - CASL IIプログラム入力して，実際に動かしてみよう
  - Web版シミュレータ
    http://www.officedaytime.com/dcaslj/
  - CaslBuilder
    http://hyamag1979.wp.xdomain.jp/soft/caslbuilder/
  - EduCasl
    http://www.vector.co.jp/soft/winnt/prog/se369290.html

COMET IIのCPU

- 汎用レジスタ(GRn)
  - GR0～GR7
  - GR1～GR7は，アドレス修飾(配列の添字のようなもの)に使える
- フラグレジスタ(FR)
  - ゼロフラグ(ZF)
  - 符号フラグ(SF)
  - オーバーフローフラグ(OF)
- スタックポインタ(SP)
- プログラムレジスタ(PR)
  - プログラムカウンタ

アセンブリ言語の例（算術演算）

```
PROG1  START MAIN          ; プログラム名と開始アドレス
; program                  ; コメント
MAIN   LD    GR0,A         ; A番地の内容をGR0に読み込む（load）
       LD    GR1,B         ; B番地の内容をGR1に読み込む
       SUBA  GR0,GR1       ; GR0の値をGR1の値だけ減らす（subtract）
       ST    GR0,X         ; GR0の値をX番地に保存（store）
       RET                 ; 実行終了（return）
; data                     ; コメント
A      DC    15            ; 定数データA：定数15（data constant）
B      DC    6             ; 定数データB：定数6
X      DS    1             ; データ領域X：1ワード分（data storage）
       END                 ; プログラム終了
```

CASL IIの命令（抜粋）

命令	動作
LD GRn,データ	レジスタGRnにデータ（レジスタGRnまたはラベル）の内容を読み込む
ST GRn,データ	レジスタGRnの内容をデータ（レジスタGRnまたはラベル）に書き込む
LAD GRn,整数	レジスタGRnに整数を直接読み込む
ADDA GRn,データ	レジスタGRnにデータの内容を足す（GRnは変化する）
SUBA GRn,データ	レジスタGRnからデータの内容を引く（GRnは変化する）
CPA GRn,DATA	GRnからデータの内容を引いたと想定し，フラグだけ変化させる（比較）
JUMP ラベル	実行番地をラベルに移す（プログラムレジスタにラベルの番地を代入）
JZE ラベル	直前の計算結果がゼロか等しいならラベルにジャンプする
JNZ ラベル	直前の計算結果がゼロでなければラベルにジャンプする
JPL ラベル	直前の計算結果がプラスならばラベルにジャンプする
JMI ラベル	直前の計算結果がマイナスならばラベルにジャンプする
CALL ラベル	次の実行番地をスタックにプッシュ（退避）して，ラベルにジャンプする
RET	スタックから実行番地をポップ（復帰）して，そこにジャンプする（戻る）
DC 整数	擬似命令：定数領域を確保して初期値を設定する
DS 整数	擬似命令：整数個分のデータ領域を確保する

条件分岐

アセンブラプログラム

```
PROG2   START   MAIN
; data
DATA1   DC 5
DATA2   DC 5
ANS     DS 1
; program
MAIN    LD      GR0,DATA1
        CPA     GR0,DATA2
        JZE     SKIP1
        LAD     GR0,0
        JUMP    SKIP2
SKIP1   LAD     GR0,1
SKIP2   ST      GR0,ANS
        RET
        END
```

ほぼ同じ意味のCプログラム

```
int data1 = 5;
int data2 = 5;
int ans;

int main(void)
{
   int gr0;
   gr0 = data1;
   if (gr0 != data2) {
        gr0 = 0;
   } else {
        gr0 = 1;
   }
   ans = gr0;
   return 0;
}
```

繰り返し

アセンブラプログラム

```
PROG4   START   MAIN
; program
MAIN    XOR     GR0,GR0
        LD      GR1,N
        LAD     GR2,1
LOOP    ADDA    GR0,GR1
        SUBA    GR1,GR2
        JNZ     LOOP
        ST      GR0,SUM
        RET
; data
N       DC      10
SUM     DS      1
        END
```

ほぼ同じ意味のCプログラム

```
int n = 10;
int sum;

int main(void)
{
   int gr0, gr1;
   gr0 = 0;  /* gr0 ^= gr0 */
   gr1 = n;
   gr2 = 1;
   do {
        gr0 += gr1;
        gr1 -= gr2;
   } while (gr1 != 0);
   sum = gr0;
   return 0;
}
```

サブルーチン(関数)

アセンブラプログラム

```
PROG5   START   MAIN
DATA1   DC      5
DATA2   DC      10
ANS     DS      1
; main routine
MAIN    LD      GR0,DATA1
        LD      GR1,DATA2
        CALL    MAX
        ST      GR0,ANS
        RET
; subroutine
MAX     CPA     GR0,GR1
        JPL     MAXR
        LD      GR0,GR1
MAXR    RET
        END
```

ほぼ同じ意味のCプログラム

```
int data1 = 5, data2 = 10;
int ans;

int main(void)
{
   ans = max(data1, data2);
   return 0;
}

int max (int gr0, int gr1)
{
   if (gr0 <= gr1) {
        gr0 = gr1;
   }
   return gr0;
}
```

プロセス管理

- プロセス(process)とは
  - 起動して“実行中”のプログラム
  - コンピュータの中で“動いているもの”(CPUを使っているもの)
  - OSによっては,同じ意味で「タスク」(task)ともいう
- OSによるプロセスの管理
  - プロセスの生成(プログラムの開始とメモリ確保)
  - プロセスの消滅(プログラムの停止とメモリ解放)
  - プロセスの切り替え,優先順位の管理,などなど…
- プロセスの“正体”を,よりハードウェア的に考えると…
  - 「動いている」or「実行中」とは = CPUで演算処理をしている
  - 「プログラムとデータがある」とは = メモリの領域を占めている

スレッド

- スレッド(thread)とは
  - 専用のメモリ空間を持たない“軽いプロセス”のこと
  - 独自の実行コンテクストを持つが，メモリは親のプロセスと共有する
  - 組み込みOSの場合，プロセスは実質的にスレッドの場合が多い
- スレッドの用途
  - 1つのプロセスの中に，さらに別々に“動くもの”を作ることができる
  - 複数の処理の並列実行によってプログラムを高速化する
  - メインの処理の“裏”でユーザやネットワークからの入力に対応する
- プロセスに比べた利点
  - 新規生成や切り替え(コンテクストスイッチ)の処理が速い
  - メモリ処理がないので，OSによる管理が単純化できる

HOSプログラミング入門

```
/* インクルードファイル(API関係の関数宣言を含む)を読み込む */
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"
#include "main.h"

/* HOS本体の起動(ここの部分は変更しない) */
int main()
{
    /* HOS-V4 の開始 */
    sta_hos(); /* ← OSなので常駐してバッググラウンドで動きづづける */
    return 0;
}
```

HOSプログラミング入門

```
/* プログラムとして実行するタスクの「関数プロトタイプ宣言」 */
void task1(VP_INT exinf); /* ← こういう関数を後ろで定義しているよ */

/* サービスコール(OSのAPI関数)cre_tskに渡すために,マニュアルに従って
   構造体T_CTSKの変数として,自作プログラムtask1の情報を設定する */
/* タスク情報: 優先度5,スタック256,拡張情報0x1234,登録後起動 */
const T_CTSK ctsk1 =
    { TA_HLNG | TA_ACT, 0x1234, task1, 5, 256, NULL };

/* HOS本体の起動後に,最初に自動的に実行される処理 */
void start(VP_INT exinf)
{
    /* サービスコールcre_tskによって,ctsk1変数で指定したタスクを起動 */
    cre_tsk(1, &ctsk1);  /* タスクIDは1に設定 */
}
```

HOSプログラミング入門

```
/* 自作したプログラムtask1 */
void task1(VP_INT exinf)
{
    SYSTIM st; /* 時間を格納する変数 */

    /* 無限に繰り返す */
    for (;;) {
        /* サービスコールget_timを用いて,現在時刻取得し表示する */
        get_tim(&st);
        printf("time: %.1lf sec¥n", (double)st.ltime / 1000);

        /* サービスコールdly_tskによって,一定時間処理を休止させる */
        dly_tsk(300);
    }
}
```

演習課題

- 課題 4a アセンブラの基礎
  - この問題の狙いは，CPUの命令実行の基礎を理解することである。
  - ノイマン型コンピュータでは，コンピュータの実行手順は「機械語」と呼ばれる2進数の命令で表され，メインメモリ（主記憶）に格納される。
  - この機械語に1対1に対応し，人間が覚えやすい名前をつけたものがアセンブリ言語（アセンブラ）であり，アセンブラを用いるとCPUを直接制御する最も低レベルなプログラミングが可能である。
  - ここでは，情報処理技術者試験で採用されているCASL IIを用いて，アセンブラのプログラミングからCPUの仕組みの基礎を学ぶ。

- 問題： 下記のCASL IIプログラムを作成し，実行結果も示せ。「コンピュータアーキテクチャ」未履修者は（1）だけでもよい。
  - （1） DCで確保した3つの整数A，B，Cを合計し，結果を保存する
  - （2） 100以上200以下の偶数を合計し，結果を保存する

演習課題

- 課題 4b HOSにおけるタスク（プロセス）の実行
  - この課題の狙いは，タスクが1つだけのプログラムを実行し，タスクを使ったプログラミングの基本を理解することである。
  - 一般にOSは，タスクの生成・開始・終了・休止等の機能を備えており，それらの処理のためのAPI（関数）を提供している。
  - 組み込みOSであるHOSでは，タスクは一関数として作成され，OSの中のスレッドとして動作する（独自のメモリを持たない）。
- 手順
  - フォルダ hos-v4¥sample¥win-task の内容を確認する。
  - sample.sln を開いてコンパイル・実行する。
  - 出力結果を示し，どうしてそうなるのか考察（理由を説明）せよ。
  - 特に，dly_tskはどのような処理を行っているのか調べてみよ。
  - 「μITRON仕様書」を参考に処理内容を考えるとよい。

HOS(μITRON)のタスク管理

サービスコール	意味	説明
cre_tsk	create task	タスクの生成
acre_tsk		同上(ID自動割り当て)
act_tsk	activate task	タスクの起動
iact_tsk		同上(割り込み用)
can_act	cancel activation	タスク起動予約の取り消し
ext_tsk	exit task	自タスクの終了
ter_tsk	terminate task	タスクの強制終了
chg_pri	change priority	タスク優先度の変更
get_pri	get priority	タスク優先度の参照
rot_rdq	rotate ready queue	実行可能キューの回転
irot_rdq		同上(割り込み用)

HOS(μITRON)のタスク管理

サービスコール	意味	説明
dly_tsk	delay task	自タスクの遅延(指定時間休止)
slp_tsk	sleep task	自タスクの休止(起床要求待ち)
tslp_tsk		同上(タイムアウトあり)
wup_tsk	wake up task	タスク起床要求
iwup_tsk		同上(割り込み用)
can_wup	cancel wakeup	タスク起床要求の取り消し
rel_wai	release wait	待ち状態の強制解除
irel_wai		同上(割り込み用)
sus_tsk	suspend task	強制待ち状態への以降
rsm_tsk	resume task	強制待ち状態からの再開
frsm_tsk	force resume task	強制待ち状態からの強制再開