２００１年９月６日

カーネル勉強会資料

～　同期通信　～

株式会社ヴィッツ　水野智仁
mizuno@witz-inc.co.jp

１．概要

同期通信機能について解説を行う。同期通信機能はタスクとは独立したオブジェクトにより、タスク間の同期通信を行うための機能である。セマフォ、イベントフラグ、データキュー、メールボックスの各機能が含まれる。
今回の資料作成に当たり、eventflag.c, eventflag.h, mailbox.c, mailbox.h, semaphore.c, semaphore.h, wait.c, wait.h, check.h, μITRON4.0 仕様書を元に作成しました。

２．セマフォ

概略

セマフォは、使用されていない資源の有無や数量を数値で表現することにより、その資源を使用する際の排他制御や同期を行うためのオブジェクトである。

セマフォ機能には以下の機能が含まれている。

- **セマフォを作成／削除する機能**
- **セマフォの資源を獲得／返却する機能**
- **セマフォの状態を参照する機能**

セマフォは、対応する資源の有無や数量を表現する資源数と、資源の獲得を待つタスクの待ち行列を持つ。資源を返却する側（イベントを知らせる側）では、セマフォの資源数を 1 つ増やす。一方、資源を獲得する側（イベントを待つ側）では、セマフォの資源数を１つ減らす。

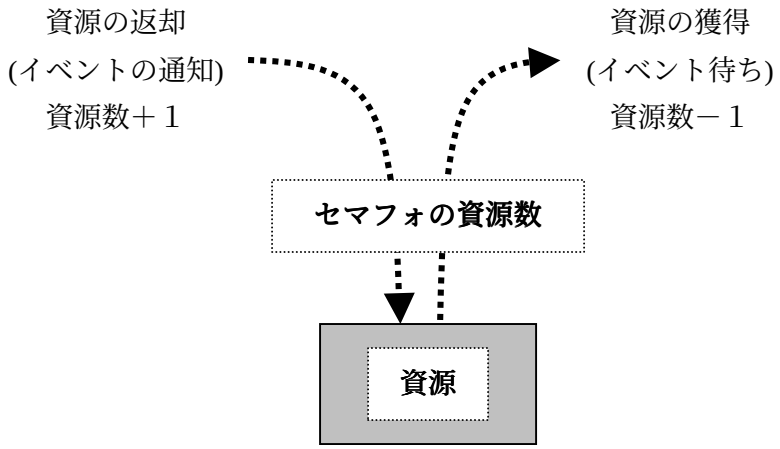


図.1　セマフォ資源遷移図

・ セマフォに対して資源が返却されすぎるのを防ぐため、最大数(TMAX_MAXSEM)を設定することができる。セマフォの最大資源数は、UINT 型（unsigned int 型に定義している）で表現できる数値の範囲内である。すなわち、unsigned int 型が 32 ビットの場合は (2^32 - 1)、16 ビットの場合は (2^16 - 1) = 65535 である。スタンダードプロファイルでは、最大数を定義する場合、65535 以上の値を指定できなければならない。JSP カーネルでは TMAX_MAXSEM は定義していない。
・ セマフォの資源が足りなくなった場合、次に資源が返却されるまで、資源を獲得しようとしたタスクはセマフォ獲得待ち状態となる。

構造体

```
typedef struct semaphore_initialization_block {
        ATR     sematr;         /* セマフォ属性 */
        UINT    isemcnt;        /* セマフォの資源数の初期値 */
        UINT    maxsem;         /* セマフォの最大資源数 */
} SEMINIB;
```

セマフォ初期化ブロック：セマフォの初期構成情報、属性･初期資源数･最大資源数。

```
typedef struct semaphore_control_block {
        QUEUE wait_queue;       /* セマフォ待ちキュー */
        const SEMINIB *seminib;/* セマフォ初期化ブロックへのポインタ */
        UINT    semcnt;         /* セマフォ現在カウント値 */
} SEMCB;
```

セマフォ管理ブロック：セマフォの管理情報。

関数

void semaphore_initialize(void)

セマフォ機能の初期化。

void sig_sem(ID semid)

semid で指定されるセマフォに対して、資源を一つ返却する。

void isig_sem(ID semid)

semid で指定されるセマフォに対して、資源を一つ返却する。非タスクコンテキスト用の

サービスコールである。

SYSCALL ER wai_sem(ID semid)

semid で指定されるセマフォから、資源を一つ獲得する。

SYSCALL ER pol_sem(ID semid)

semid で指定されるセマフォから、資源を一つ獲得する。wai_sem の処理をポーリングで行うサービスコール。

SYSCALL ER twai_sem(ID semid, TMO tmout)

semid で指定されるセマフォから、資源を一つ獲得する。wai_sem にタイムアウト機能を付け加えたの処理をポーリングで行うサービスコール。

３．イベントフラグ

概略

イベントフラグは、イベントの有無をビット毎のフラグで表現することにより、同期を行うためのオブジェクトである。

イベントフラグ機能では以下の機能が含まれている。

- **イベントフラグを生成／削除する機能**
- **イベントフラグをセット／クリアする機能**
- **イベントフラグで待つ機能**
- **イベントフラグの状態を参照する機能**

イベントを知らせる側では、イベントフラグのビットパターンの指定したビットをセットないしはクリアする事が可能である。一方、イベントを待つ側では、イベントフラグのビットパターンの指定したビットの全て又はいずれかがセットされるまで、タスクをイベントフラグ待ち状態にする事が出来る。

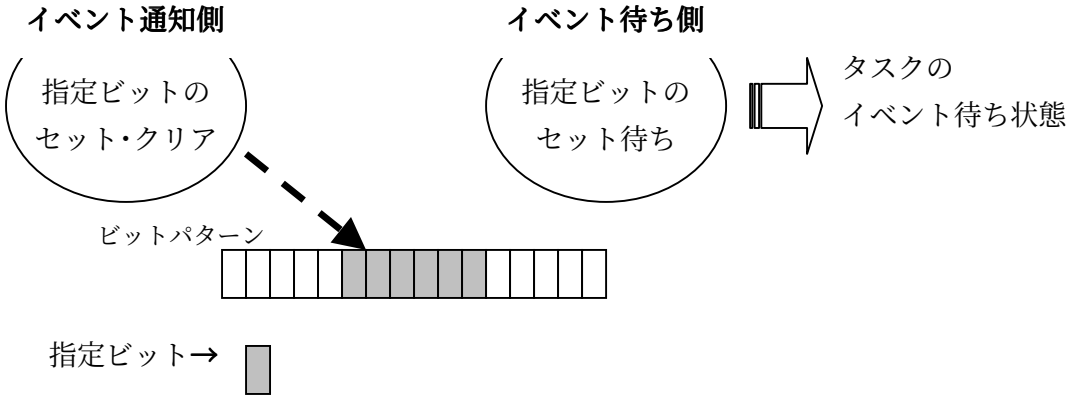


図.2　イベントフラグ状態図

- イベントフラグ属性に TA_WMUL を指定すれば、一つのイベントフラグで複数のタスクが待ち状態になる機能があるが、JSP カーネルでは、イベントフラグで複数のタスクが待ち状態になれる機能はサポートしていない。

構造体

```
typedef struct eventflag_waiting_information {
        WINFO           winfo;   /* 標準の待ち情報ブロック */
        WOBJCB          *wobjcb;/* 待ちオブジェクトのコントロールブロック */
        FLGPTN          waiptn; /* 待ちパターン */
        MODE            wfmode;/* 待ちモード */
        FLGPTN          flgptn;  /* 待ち解除時のパターン */
} WINFO_FLG;
```

イベントフラグ待ち情報ブロックの定義

```
typedef struct eventflag_initialization_block {
        ATR             flgatr;  /* イベントフラグ属性 */
        FLGPTN          iflgptn; /* イベントフラグのビットパターンの初期値 */
} FLGINIB;
```

イベントフラグ初期化ブロック

```
typedef struct eventflag_control_block {
        QUEUE wait_queue;    /* イベントフラグ待ちキュー */
        const FLGINIB *flginib;/* イベントフラグ初期化ブロックへのポインタ */
        FLGPTN          flgptn;/* イベントフラグ現在パターン */
} FLGCB;
```

イベントフラグ管理ブロック

関数

void eventflag_initialize(void)

イベントフラグ機能の初期化

static BOOL eventflag_cond(FLGCB *flgcb, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)

イベントフラグ待ち解除条件のチェック

SYSCALL ER set_flg(ID flgid, FLGPTN setptn)

flgid で指定されるイベントフラグに対して、setptn で指定されるビットをセットする。

SYSCALL ER iset_flg(ID flgid, FLGPTN setptn)

flgid で指定されるイベントフラグに対して、setptn で指定されるビットをセットする。非タスクコンテキスト用のサービスコールである。

SYSCALL ER clr_flg(ID flgid, FLGPTN clrptn)

flgid で指定されるイベントフラグに対して、clrptn の対応するビットが 0 になっているビットをクリアする。

SYSCALL ER wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)

flgid で指定されるイベントフラグのビットパターンが、waiptn と wfmode で指定される待ち解除条件を満たすのを待つ。

SYSCALL ER pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)

flgid で指定されるイベントフラグのビットパターンが、waiptn と wfmode で指定される待ち解除条件を満たすのを待つ。また、waiptn の処理をポーリングで行うサービスコールである。

SYSCALL ER twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout)

flgid で指定されるイベントフラグのビットパターンが、waiptn と wfmode で指定される待ち解除条件を満たすのを待つ。また、twai_flg は wai_flg にタイムアウトの機能を付け加えたサービスコールである。

４．メールボックス

概略

メールボックスは、共有メモリ上に置かれたメッセージを受渡しすることにより、同期と通信を行うためのオブジェクトである。

メールボックス機能には以下の機能が含まれている。

- **メールボックスを生成／削除する機能**
- **メッセージを送信／受信する機能**
- **メールボックスの状態を参照する機能**

メールボックスは、送信されたメッセージを入れるためのメッセージキューと、メッセージの受信を待つタスクの待ち行列を持つ。メッセージを送信する側(イベントを知らせる側)では、送信したいメッセージをメッセージキューに入れる。一方、メッセージを受信する側(イベントを待つ側)では、メッセージキューに入っているメッセージを一つ取り出す。

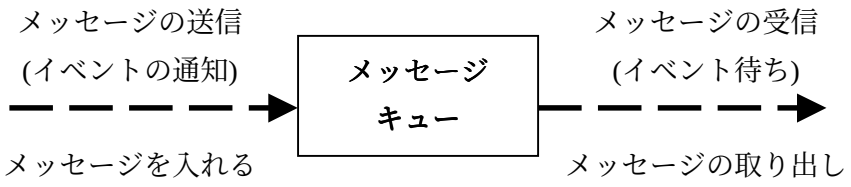
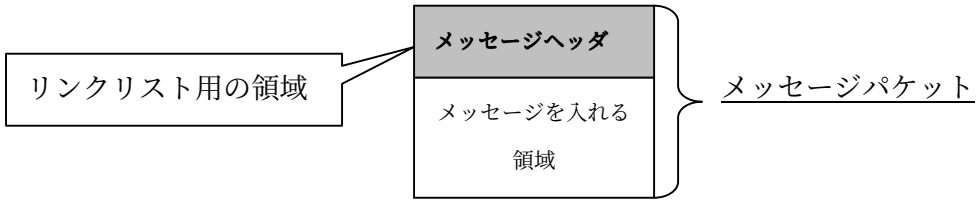


図.3　ﾒｯｾｰｼﾞｷｭｰ遷移図

- メッセージキューにメッセージが入っていない場合は、次に送られてくるまでメールボックスからの受信待ち状態になる。
- メールボックスによって実際に送受信されるのは、送信側と受信側で共有しているメモリ上に置かれたメッセージの先頭番地のみである。すなわち、送受信されるメッセージの内容のコピーは行われない。

カーネルは、メッセージキューに入っているメッセージを、リンクリストにより管理する。アプリケーションプログラムは、送信するメッセージの先頭に、カーネルがリンクリストに用いるための領域を確保しなければならない。この領域をメッセージヘッダと呼ぶ。また、メッセージヘッダと、それに続きアプリケーションがメッセージを入れるための領域を合わせて、メッセージパケットと呼ぶ。



メッセージの構成

図.4　メッセージパケット構成図

構造体

```
typedef  struct t_msg {
        struct t_msg      *next;
} T_MSG;
```

メールボックスのメッセージヘッダ

```
typedef struct mailbox_waiting_information {
        WINFO           winfo;    /* 標準の待ち情報ブロック */
        WOBJCB          *wobjcb; /* 待ちオブジェクトのコントロールブロック */
        T_MSG           *pk_msg;/* 受信したメッセージ */
} WINFO_MBX;
```

メールボックス待ち情報ブロックの定義

```
typedef struct mailbox_initialization_block {
        ATR     mbxatr;         /* メールボックス属性 */
        PRI     maxmpri;        /* メッセージ優先度の最大値 */
} MBXINIB;
```

メールボックス初期化ブロック

```
typedef struct mailbox_control_block {
        QUEUE wait_queue;       /* メールボックス待ちキュー */
        const MBXINIB *mbxinib;/* メールボックス初期化ブロックへのポインタ */
        T_MSG *head;            /* 先頭のメッセージ */
        T_MSG *last;            /* 末尾のメッセージ */
} MBXCB;
```

メールボックス管理ブロック

関数

void mailbox_initialize(void)

メールボックス機能の初期化。キューの初期化、メールボックス初期化ブロックエリアの初期化などを行っている。

Inline void enqueue_msg_pri(T_MSG **p_prevmsg_next, T_MSG *pk_msg)

優先度順位メッセージキューへの挿入。

SYSCALL ER snd_mbx(ID mbxid, T_MSG *pk_msg)

mbxid で指定されるメールボックスに、pk_msg を先頭番地とするメッセージを送信する。具体的な処理では、タスク待ち状態の解除、メッセージ属性の変更(TA_MFIFO,TA_MPRI)を行っている。

SYSCALL ER rcv_mbx(ID mbxid, T_MSG **ppk_msg)

mbxid で指定されるメールボックスからメッセージを受信し、その先頭番地を pk_msg に返す。メッセージが入っていない場合は、受信待ち状態へ移行する。

SYSCALL ER prcv_mbx(ID mbxid, T_MSG **ppk_msg)

mbxid で指定されるメールボックスからメッセージを受信し、その先頭番地を pk_msg に返す。メッセージが入っていない場合は、E_TMOUT を返す。

SYSCALL ER trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout)

mbxid で指定されるメールボックスからメッセージを受信し、その先頭番地を pk_msg に返す。メッセージが入っていない場合は、受信待ち状態への移行、又は, E_TMOUT を返す。

５．同期通信オブジェクトに対する待ち状態

タスクが待ち状態の間は、TCB 及びそこから指される WINFO を次のように設定しなければならない。

待ち情報ブロックの定義

```
typedef union waiting_information {
        ER          wercd;      /* 待ち解除時のエラーコード */
        TMEVTB      *tmevtb;    /* 待ち状態用のタイムイベントブロック */
} WINFO;
```

```
typedef struct time_event_block {
        UINT    index;          /* タイムイベントヒープ中での位置 */
        CBACK callback;         /* コールバック関数 */
        VP      arg;            /* コールバック関数へ渡す引数 */
} TMEVTB;
```

待ち状態を解除する際には、待ち解除したタスクからの返値を WINFO の wercd に設定する。

タイムアウトを監視するために、待ち状態用のタイムイベントブロックを登録する。登録するタイムイベントブロックは、待ちに入るサービスコール処理関数のローカル変数として確保し、それへのポインタを WINFO の tmevtb に記憶する。タイムアウトの監視が必要ない場合（永久待ちの場合）には、tmevtb を NULL にする。

wercd を使うのは待ち解除以降であるのに対して、tmevtb は待ち解除後は使わないため、メモリ節約のために共用体（union）を使っている。

同期・通信オブジェクト待ち情報ブロックの定義

標準の WINFO に wobjcb フィールドを追加した下記の構造体を使い、タスク状態に TS_WAIT_WOBJ を設定する。

```
typedef struct wait_object_waiting_information {
        WINFO           winfo;   /* 標準の待ち情報ブロック */
        WOBJCB          *wobjcb;/* 待ちオブジェクトのコントロールブロック */
} WINFO_WOBJ;
```

待ち対象の同期・通信オブジェクトのコントロールブロックへのポインタを、WINFO_WOBJ の wobjcb に記憶する。

JSP カーネルで wobjcb を使うのは，優先度順の待ちキューにつながれているタスクの優先度が変更された場合のみであるが、デバッグ情報を取り出しやすいように、待ちキューが優先度順でない場合にも wobjcb を設定する。また、待ち対象の同期・通信オブジェクトに依存して記憶することが必要な情報がある場合には、WINFO_WOBJ に必要な情報のためのフィールドを追加した構造体を定義し、WINFO_WOBJ の代わりに用いる。

構造体

同期・通信オブジェクトの初期化ブロックの共通部分

```
typedef struct wait_object_initialization_block {
        ATR     wobjatr; /*  オブジェクト属性  */
} WOBJINIB;
```

同期・通信オブジェクトのコントロールブロックの共通部分

```
typedef struct wait_object_control_block {
        QUEUE           wait_queue;         /*  待ちキュー  */
        const WOBJINIB          *wobjinib; /*  初期化ブロックへのポインタ  */
} WOBJCB;
```

関数

Inline void queue_insert_tpri(TCB *tcb, QUEUE *queue)
タスクの優先度順の待ちキューへの挿入。

Inline void wobj_queue_insert(WOBJCB *wobjcb)
実行中のタスクの同期通信オブジェクトの待ちキューへの挿入。

Void wobj_make_wait(WOBJCB *wobjcb, WINFO_WOBJ *winfo)
同期通信オブジェクトに対する待ち状態への移行。

Void wobj_make_wait_tmout(WOBJCB *wobjcb, WINFO_WOBJ *winfo, TMEVTB *tmevtb, TMO tmout)
同期通信オブジェクトに対する待ち状態への移行。タイムアウト指定。

Void wobj_change_priority(WOBJCB *wobjcb, TCB *tcb)
タスクの優先度変更時の処理。

６．マクロ

エラーチェック一覧

セマフォ、イベントフラグ、メールボックスの中にて使用されているパラメータチェック用の各マクロについて記述する。

エラーチェック名称	説明
CHECK_TSKCTX_UNL()	呼出しコンテキストと CPU ロック状態のチェック。現在のコンテキストが非タスクコンテキスト、CPU ロック状態のいずれかの状態の時、E_CTX(コンテキストエラー)を返す。
CHECK_SEMID(semid)	オブジェクト ID のチェック。semid が TMIN_SEMID から tmax_semid の範囲外ならば、E_ID(不正 ID 番号)を返す。
CHECK_INTCTX_UNL()	呼出しコンテキストと CPU ロック状態のチェック。現在のコンテキストがタスクコンテキスト、CPU ロック状態のいずれかの状態の時、E_CTX(コンテキストエラー)を返す。
CHECK_DISPATCH()	ディスパッチ保留状態でないかのチェック。現在のコンテキストが非タスクコンテキスト、CPU ロック状態、ディスパッチ禁止状態のいずれかの状態の時、E_CTX(コンテキストエラー)を返す。
CHECK_TMOUT(tmout)	タイムアウト指定値のチェック。tmout の値が TMO_FEVR(-1:永久待ち)より負の値の場合、E_PAR(パラメータエラー)を返す。
CHECK_FLGID(flgid)	オブジェクト ID のチェック。flgid が TMIN_FLGID から tmax_flgid の範囲外ならば、E_ID(不正 ID 番号)を返す。
CHECK_PAR(exp)	その他のパラメータエラーのチェック。exp の値が NULL(0) ならば、E_PAR(パラメータエラー)を返す。
CHECK_MBXID(mbxid)	オブジェクト ID のチェック。mbxid が TMIN_MBXID から tmax_mbxid の範囲外ならば、E_ID(不正 ID 番号)を返す。

マクロ一覧

セマフォ、イベントフラグ、メールボックスの中にて使用されているマクロについて記述する。

マクロ名	機能説明
TMO_POL	タイムアウト指定：ポーリング
TWF_ORW	サービスコールの動作モード：イベントフラグの or 待ちを指定する。
TMIN_MPRI	メッセージ優先度の最小値
TS_WAITING	タスク状態を『待ち状態』に指定。task.h に記述。
TA_WAIT_WOBJ	同期通信オブジェクトに対する待ち状態.。
TA_WAIT_WOBJCB	共通部分の待ちキューにつながっている
TA_CLR	オブジェクト属性を『待ち解除時にイベントフラグをクリア』に指定する。
TA_WMUL	オブジェクト属性を『イベントを複数のタスクが持つことを許す』に指定する。
TA_MPRI	オブジェクト属性を『メッセージのキューをメッセージの優先度順』に指定する。
TA_TFIFO	タスク待ち行列を FIFO 順に
TA_TPRI	タスク待ち行列をタスク優先度順に
E_ILUSE	サービスコール不正使用。サービスコールを呼び出したコンテキストや操作対象のオブジェクトの状態に依存して発生する。
E_QOVR	キューイングオーバフロー:キューイング又はネスト可能なサービスコールで、その上限回数を超えた事を示すエラー。
E_TMOUT	ポーリング失敗、又は、タイムアウト。タイムアウトが発生した、又はポーリングに失敗した事を示すエラー。
TMIN_MBXID	メールボックス ID の最小値
TMIN_FLGID	フラグ ID の最小値
TMIN_SEMID	セマフォ ID の最小値