Embedded Multicore Consortium

www.embeddedmulticore.org

マルチコアソフト開発実践編

～並列処理の不具合と対策～

2021.11

ガイオテクノロジー(株)
浅野昌尚

Embedded Multicore Consortium

並列処理プログラムの不具合

■ 昨年は並列処理プログラミング

掛け算プログラムを複数の方法で並列化して逐次処理性能を上回る試み
EMCサイト内ブログに情報あり

■ 今年は並列プログラムの不具合事例

加算プログラムの並列化で発生する不具合の事例

- 排他処理漏れ
- デッドロック
- ライブロック

こちらのテーマもEMCブログ更新中

連続数値の加算プログラム

■ 0～引数値までの整数を加算するプログラム

- 加算結果は4バイト
- 逐次処理関数が4バイトサイズで加算する
- 並列処理関数は2バイトサイス゛で加算する
- 結果が同じことを確認する

4バイト計算関数

```
void add_int(int num)
{
  int i,*p;

  for( i = 0, p = array ; i < num ; i++ )
    int_sum += *p++;
}
```

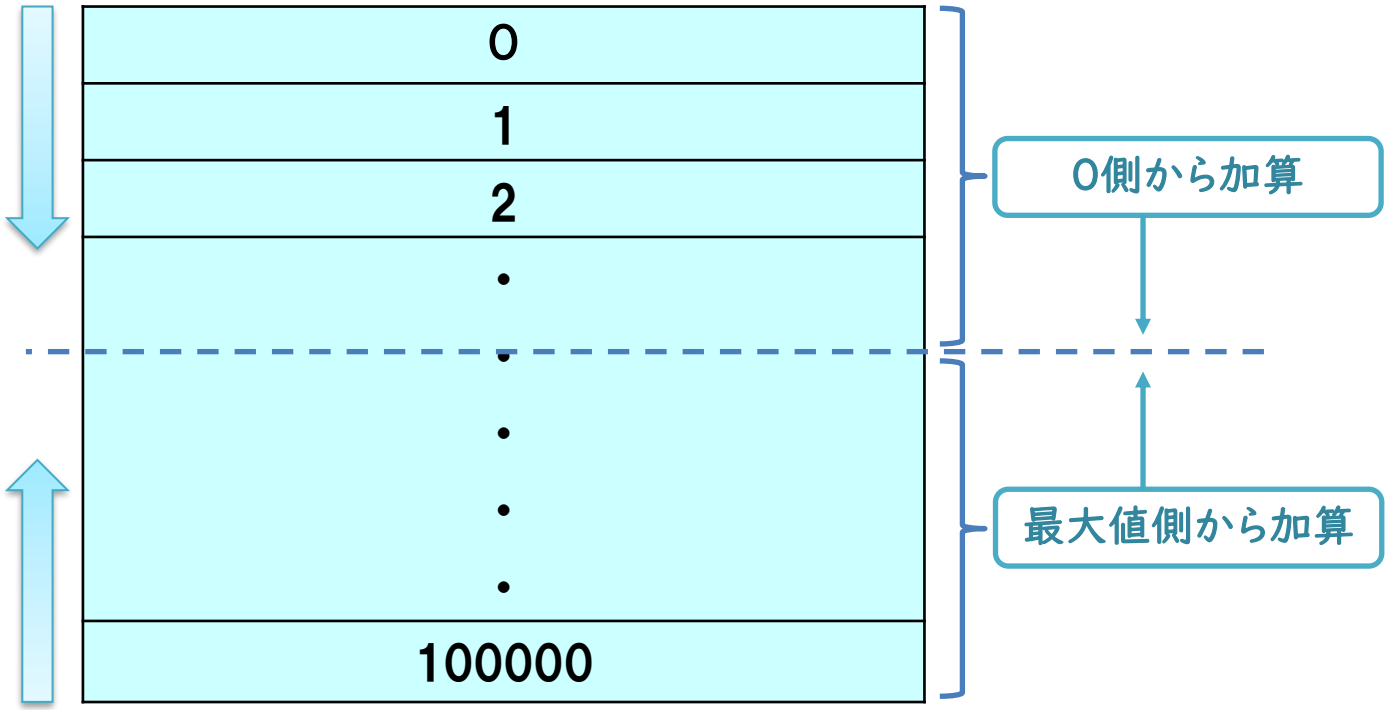
2バイト計算関数

```
void add_short(int num)
{
  int i,j,*p;
  unsigned short u,l;

  for( i = 0, p = array ; i < num ; i++ )
  {
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    p++;
  }
}
```

並列処理化

■ 2バイト関数の並列化

- 加算は4バイトデータを上下各2バイトに分けて加算する
- 2並列で処理
- 一方は上から、もう一方は下から加算してゆく
- ぶつかったところで停止



並列処理化

■ 2バイト関数の並列加算プログラム

小さい方から加算

```
void add_short_top(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = 0, p = array ; i < num ; i++ )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    p++;
  }
  pthread_exit(NULL);
}
```

下位、上位順で加算

大きい方から加算

```
void add_short_end(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; i-- )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    p--;
  }
  pthread_exit(NULL);
}
```

上位、下位順で加算

動作確認 → 誤動作

■ 動作例

- 並列処理では正しく加算できない

逐次処理の場合

```
$ ./sample1_0.exe 100000
 Count = 100000
OK : 704982704
```

並列処理の場合

（2バイト加算を並列処理）

```
$ ./sample1_1.exe 100000
 Count = 100000
NG : 704982704(int) 3953354080(short)
```

並列処理の結果は
デタラメ

誤動作の原因

■ 並列化失敗原因は競合

- 2並列で動作する関数が同じ変数をアクセスしてしまう問題
- 読むだけなら構わないが、書き込むと問題が発生する

小さい方から加算

```
void add_short_top(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = 0, p = array ; i < num ; i++ )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    p++;
  }
  pthread_exit(NULL);
}
```

大きい方から加算

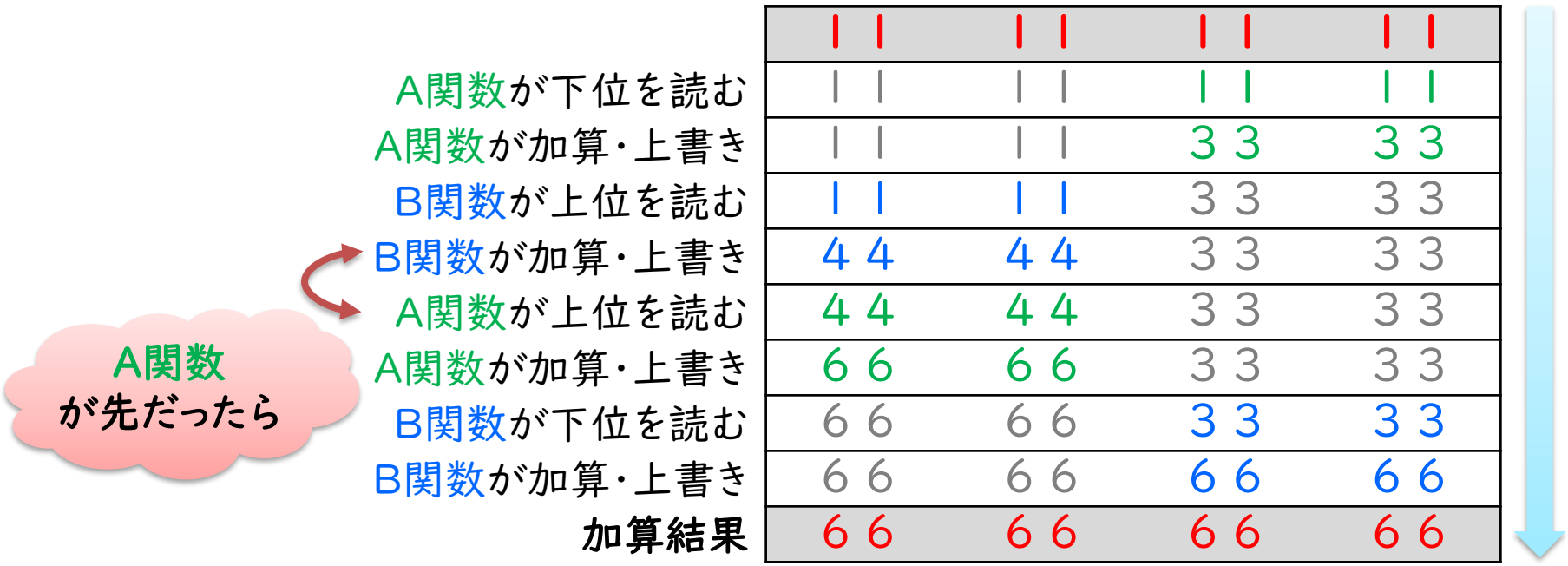
```
void add_short_end(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; i-- )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    p--;
  }
  pthread_exit(NULL);
}
```

short_sum_u と short_sum_l は2つの関数が使用しており、
両関数が読み書きする

競合による誤動作

■ 並列動作で何が起きているのか

0x11111111に、A関数が0x22222222を、B関数が0x33333333を加算



競合による誤動作

■ 一部が入れ替わると正しい結果を得られない

0x11111111に、A関数が0x22222222を、B関数が0x33333333を加算

	1 1	1 1	1 1	1 1
A関数が下位を読む	1 1	1 1	1 1	1 1
A関数が加算・上書き	1 1	1 1	3 3	3 3
B関数が上位を読む	1 1	1 1	3 3	3 3
A関数が上位を読む	1 1	1 1	3 3	3 3
B関数が加算・上書き	4 4	4 4	3 3	3 3
A関数が加算・上書き	3 3	3 3	3 3	3 3
B関数が下位を読む	3 3	3 3	3 3	3 3
B関数が加算・上書き	3 3	3 3	6 6	6 6
加算結果	3 3	3 3	6 6	6 6

問題あり

競合状態の回避

■ 排他処理

- 指定区間では他関数の動作を排除
- 動作するには優先権が必要(優先権が得られなければ、待つ)

小さい方から加算

```
void add_short_top(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = 0, p = array ; i < num ; i++ )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    p++;
  }
  pthread_exit(NULL);
}
```

大きい方から加算

```
void add_short_end(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; i-- )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    p--;
  }
  pthread_exit(NULL);
}
```

この区間をどちらか片方だけが動作するように制限すれば、
ここの動作中に他の関数は動かない

排他処理

- mutex変数による排他処理(semaphoreを使う方法もある)
  - mutex_1(mutex変数)初期化後、lock/unlock制御
  - mutex_1をlockできた方が動作する(semaphoreはカウンタ制御)
  - lock発行時、既にlock中であれば、unlockされるまで待たされる

小さい方から加算

```
void add_short_top(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = 0, p = array ; i < num ; i++ )
  {
    if ( pthread_mutex_lock(&mutex_1) )   // mutex変数ロック
      printf(" Mutex Lock Error (add_short_top)¥n");
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    p++;
    if ( pthread_mutex_unlock(&mutex_1) ) // mutex変数アンロック
      printf(" Mutex Unlock Error (add_short_top)¥n");
  }
  pthread_exit(NULL);
}
```

大きい方から加算

```
void add_short_end(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; i-- )
  {
    if ( pthread_mutex_lock(&mutex_1) )   // mutex変数ロック
      printf(" Mutex Lock Error (add_short_end)¥n");
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    p--;
    if ( pthread_mutex_unlock(&mutex_1) ) // mutex変数アンロック
      printf(" Mutex Unlock Error (add_short_end)¥n");
  }
  pthread_exit(NULL);
}
```

排他区間を絞り込む

- 排他区間を必要最小限に
- mutex変数をリソース毎に割り当ててみる
  - mutex_u(short_sum_u用)とmutex_l(short_sum_l用)

```
#define Lock_U if(pthread_mutex_lock(&mutex_u)) printf(" Mutex Lock Error (mutex_u)¥n")
#define Lock_L if(pthread_mutex_lock(&mutex_l)) printf(" Mutex Lock Error (mutex_l)¥n")
#define Unlock_U if(pthread_mutex_unlock(&mutex_u)) printf(" Mutex Unlock Error (mutex_u)¥n")
#define Unlock_L if(pthread_mutex_unlock(&mutex_l)) printf(" Mutex Unlock Error (mutex_l)¥n")
```

```
void add_short_top(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = 0, p = array ; i < num ; i++ )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    Lock_L;                    // Lock L
    Lock_U;                    // Lock U
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_L;                  // Unlock L
    Unlock_U;                  // Unlock U
    p++;
  }
  pthread_exit(NULL);
}
```

小さい方から加算

排他制御区間

```
void add_short_end(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; i-- )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    Lock_U;                    // Lock U
    Lock_L;                    // Lock L
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_U;                  // Unlock U
    Unlock_L;                  // Unlock L
    p--;
  }
  pthread_exit(NULL);
}
```

大きい方から加算

終らない

- printfにて状況を確認

```
#define Lock_U(A) if(pthread_mutex_lock(&mutex_u)) printf(" Mutex Lock Error (mutex_u)¥n"); else printf(" Lock (mutex_u) in %s¥n",A);
#define Lock_L(A) if(pthread_mutex_lock(&mutex_l)) printf(" Mutex Lock Error (mutex_l)¥n"); else printf(" Lock (mutex_l) in %s¥n",A);
#define Unlock_U(A) if(pthread_mutex_unlock(&mutex_u)) printf(" Mutex Unlock Error (mutex_u)¥n"); else printf(" UnLock (mutex_u) in %s¥n",A);
#define Unlock_L(A) if(pthread_mutex_unlock(&mutex_l)) printf(" Mutex Unlock Error (mutex_l)¥n"); else printf(" UnLock (mutex_l) in %s¥n",A);
```

```
void add_short_top(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = 0, p = array ; i < num ; i++ )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    Lock_L("top");                  // Lock L
    Lock_U("top");                  // Lock U
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_L("top");                // Unlock L
    Unlock_U("top");                // Unlock U
    p++;
  }
  pthread_exit(NULL);
}
```

小さい方から加算

排他制御区間

```
void add_short_end(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; i-- )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    Lock_U("end");                  // Lock U
    Lock_L("end");                  // Lock L
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_U("end");                // Unlock U
    Unlock_L("end");                // Unlock L
    p--;
  }
  pthread_exit(NULL);
}
```

大きい方から加算

2つ目のロック処理で停止

■ 互いに相手関数のunlock 待ち

```
$ ./sample1_3_1.exe 100000
 Count = 100000
 Lock (mutex_l) in top
 Lock (mutex_u) in end
```

```
void add_short_top(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = 0, p = array ; i < num ; i++ )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    Lock_L("top");                  // Lock L
    Lock_U("top");                  // Lock U
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_L("top");                // Unlock L
    Unlock_U("top");                // Unlock U
    p++;
  }
  pthread_exit(NULL);
}
```

停止箇所

排他制御区間

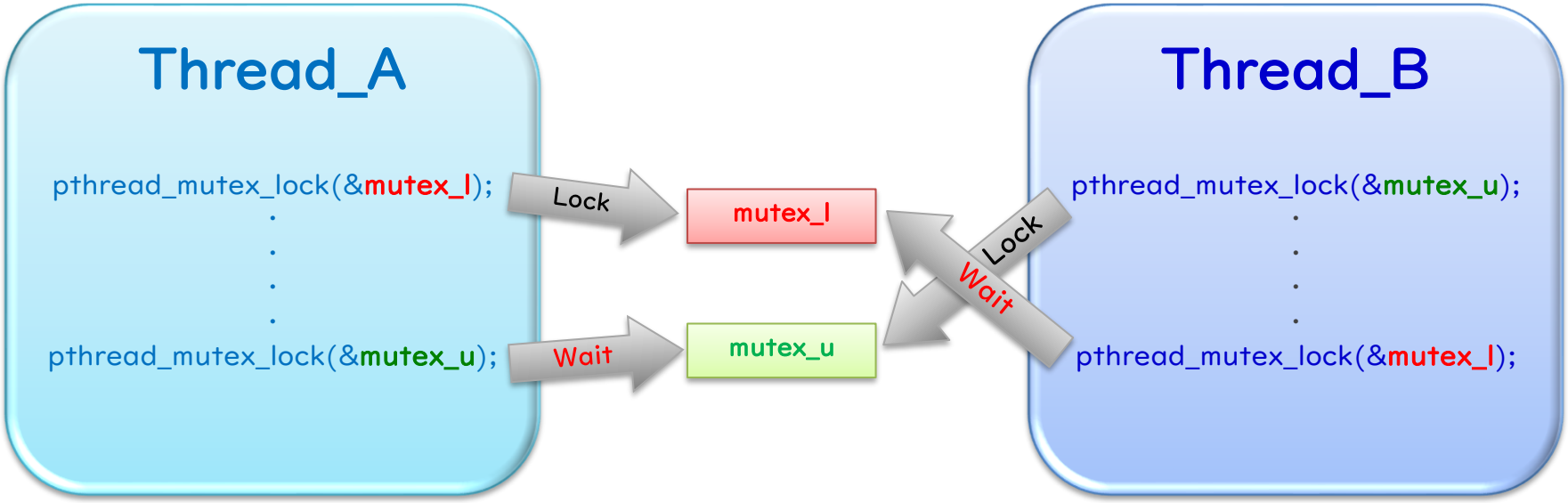
小さい方から加算

```
void add_short_end(int *nump)
{
  int num=*nump,i,j,*p;
  unsigned short u,l;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; i-- )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    Lock_U("end");                  // Lock U
    Lock_L("end");                  // Lock L
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_U("end");                // Unlock U
    Unlock_L("end");                // Unlock L
    p--;
  }
  pthread_exit(NULL);
}
```

大きい方から加算

デッドロック

2つのスレッドが異なる順序で2つのミューテックス変数をロックする際にデッドロックが発生する

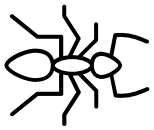


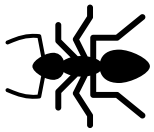
2つのスレッドが互いに相手の2番目のミューテックス変数をロックすると、処理が停止する

ロック順を揃えるとデッドロックは解消する

ライブロックとは

- 2つのスレッドが同じ回避操作を行う
  結果的にお互いが回避操作を繰り返すだけで、そこから先へ進めない状況に陥ってしまう

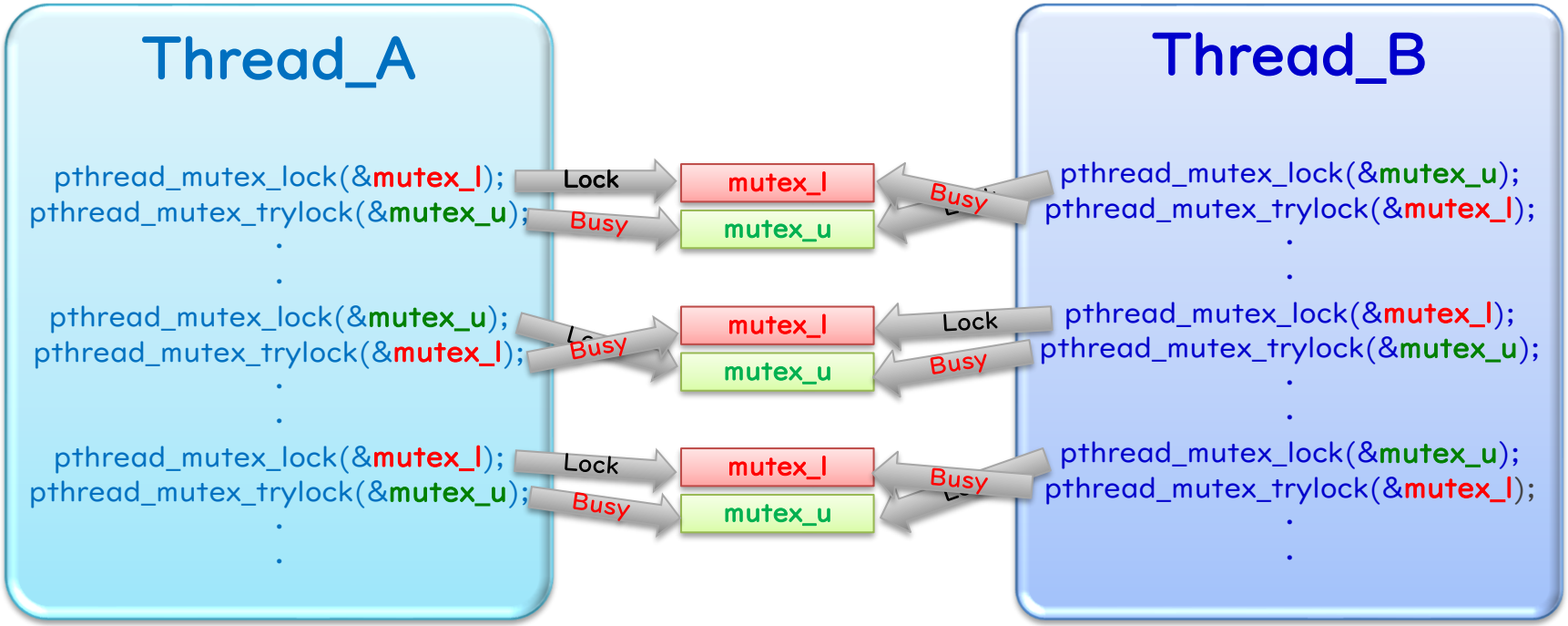




ライブロックを
発生させるのは
とても大変

ライブロックとは

- 2つのミューテックス変数の片方を抑え合う
  両関数がミューテックス変数のロック順を入れ替えても2個のロックができない状態になり、この入れ替えを永久に繰り返してしまう



両スレッド共
2つのミューテックスロックができず、とロック順を変える処理を
無限に繰り返すライブロック状態が発生する

ライブロックを起こす

```
void add_short_top(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = 0, p = array ; i < num ; )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    for ( ; ; )
    {
    Lock_P1;                        // Lock p1
    if ( TryLock_P2 == EBUSY )  // Lock p2
    {
      usleep(1);
      Unlock_P1;
      pw = p1, p1 = p2, p2 = pw;
//    printf("Busy:Top (%d)¥n",i);
      continue;
    }
    break;
    }
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_P1;                  // Unlock p1
    Unlock_P2;                  // Unlock p2
    p++;
    i++;
  }
  pthread_exit(NULL);
}
```

この区間で
回避行動を
繰り返す

```
void add_short_end(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    for ( ; ; )
    {
    Lock_P2;                        // Lock p2
    if ( TryLock_P1 == EBUSY )  // Lock p1
    {
      usleep(1);
      Unlock_P2;
      pw = p1, p1 = p2, p2 = pw;
//    printf("Busy:End (%d)¥n",i);
      continue;
    }
    break;
    }
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_P2;                  // Unlock p2
    Unlock_P1;                  // Unlock p1
    p--;
    i--;
  }
  pthread_exit(NULL);
}
```

```
#define Lock_P1 if(pthread_mutex_lock(p1)) printf(" Mutex Lock Error (p1)¥n") /* ;else  printf(" Mutex Lock (p1)¥n") */
#define Lock_P2 if(pthread_mutex_lock(p2)) printf(" Mutex Lock Error (p2)¥n") /* ;else  printf(" Mutex Lock (p2)¥n") */
#define TryLock_P1 (pthread_mutex_trylock(p1))
#define TryLock_P2 (pthread_mutex_trylock(p2))
#define Unlock_P1 if(pthread_mutex_unlock(p1)) printf(" Mutex Unlock Error (p1)¥n") /* ;else  printf(" Mutex Unlock (p1)¥n") */
#define Unlock_P2 if(pthread_mutex_unlock(p2)) printf(" Mutex Unlock Error (p2)¥n") /* ;else  printf(" Mutex Unlock (p2)¥n") */
```

ライブロックを起こす

■ とてもデリケートな発生条件

```
void add_short_top(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = 0, p = array ; i < num ; )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    for ( ; ; )
    {
    Lock_P1;                    // Lock p1
    if ( TryLock_P2 == EBUSY )  // Lock p2
    {
      usleep(1);
      Unlock_P1;
      pw = p1, p1 = p2, p2 = pw;
//    printf("Busy:Top (%d)¥n",i);
      continue;
    }
    break;
    }
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_P1;                  // Unlock p1
    Unlock_P2;                  // Unlock p2
    p++;
    i++;
  }
  pthread_exit(NULL);
}
```

ライブロックは
なかなか発生しない

例えば、
usleep(1)の削除

printf()の挿入

ライブロックの確認

- gdbによる確認(printfは使えそうにない)

```
(gdb) info thread
  Id   Target Id                         Frame
  1    Thread 15488.0x1ef0 "sample1_4" 0x00007ffb7f36d974 in ntdll!ZwWaitForMultipleObjects () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
  2    Thread 15488.0x9dc              0x00007ffb7f370874 in ntdll!ZwWaitForWorkViaWorkerFactory () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
  3    Thread 15488.0x2f48             0x00007ffb7f370874 in ntdll!ZwWaitForWorkViaWorkerFactory () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
  4    Thread 15488.0x4efc "sig"       0x00007ffb7f36cee4 in ntdll!ZwReadFile
    () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
  5    Thread 15488.0x3384             0x00007ffb7f370874 in ntdll!ZwWaitForWorkViaWorkerFactory () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
  6    Thread 15488.0x5284 "sample1_4" 0x00007ffb7f36d974 in ntdll!ZwWaitForMultipleObjects () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
  7    Thread 15488.0x3214 "sample1_4" 0x00007ffb7f36d974 in ntdll!ZwWaitForMultipleObjects () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
* 8    Thread 15488.0x3ff0             0x00007ffb7ce3d7cf in TlsGetValue ()
   from /cygdrive/c/WINDOWS/System32/KERNELBASE.dll
(gdb) thread 6
[Switching to thread 6 (Thread 15488.0x5284)]
#0  0x00007ffb7f36d974 in ntdll!ZwWaitForMultipleObjects () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
(gdb) where
#0  0x00007ffb7f36d974 in ntdll!ZwWaitForMultipleObjects () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
#1  0x00007ffb7cdccab0 in WaitForMultipleObjectsEx () from /cygdrive/c/WINDOWS/System32/KERNELBASE.dll
#2  0x00007ffb7cdcc9ae in WaitForMultipleObjects () from /cygdrive/c/WINDOWS/System32/KERNELBASE.dll
#3  0x0000000180048bbc in cygwait(void*, _LARGE_INTEGER*, unsigned int) () from /usr/bin/cygwin1.dll
#4  0x000000018014298d in clock_nanosleep () from /usr/bin/cygwin1.dll
#5  0x0000000180142dd8 in usleep () from /usr/bin/cygwin1.dll
#6  0x000000018013e96b in _sigfe () from /usr/bin/cygwin1.dll
#7  0x0000000100401222 in add_short_top (nump=0xffffcbc0) at sample1_4.c:79
#8  0x000000018016d45f in pthread::thread_init_wrapper(void*) () from /usr/bin/cygwin1.dll
#9  0x00000001800ddbba in pthread_wrapper () from /usr/bin/cygwin1.dll
#10 0x0000000000000000 in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
(gdb) thread 7
[Switching to thread 7 (Thread 15488.0x3214)]
#0  0x00007ffb7f36d974 in ntdll!ZwWaitForMultipleObjects () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
(gdb) where
#0  0x00007ffb7f36d974 in ntdll!ZwWaitForMultipleObjects () from /cygdrive/c/WINDOWS/SYSTEM32/ntdll.dll
#1  0x00007ffb7cdccab0 in WaitForMultipleObjectsEx () from /cygdrive/c/WINDOWS/System32/KERNELBASE.dll
#2  0x00007ffb7cdcc9ae in WaitForMultipleObjects () from /cygdrive/c/WINDOWS/System32/KERNELBASE.dll
#3  0x0000000180048bbc in cygwait(void*, _LARGE_INTEGER*, unsigned int) () from /usr/bin/cygwin1.dll
#4  0x000000018014298d in clock_nanosleep () from /usr/bin/cygwin1.dll
#5  0x0000000180142dd8 in usleep () from /usr/bin/cygwin1.dll
#6  0x000000018013e96b in _sigfe () from /usr/bin/cygwin1.dll
#7  0x0000000100401303 in add_short_end (nump=0xffffcbc0) at sample1_4.c:117
#8  0x000000018016d45f in pthread::thread_init_wrapper(void*) () from /usr/bin/cygwin1.dll
#9  0x00000001800ddbba in pthread_wrapper () from /usr/bin/cygwin1.dll
#10 0x0000000000000000 in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
```

停止箇所

停止箇所

ライブロックの確認

- gdbによる確認(Ctrl-Cにより停止箇所を複数回確認)

```
void add_short_top(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = 0, p = array ; i < num ; )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    for ( ; ; )
    {
      Lock_P1;                      // Lock p1
      if ( TryLock_P2 == EBUSY )    // Lock p2
      {
        usleep(1);
        Unlock_P1;
        pw = p1, p1 = p2, p2 = pw;
//      printf("Busy:Top (%d)¥n",i);
        continue;
      }
      break;
    }
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_P1;                  // Unlock p1
    Unlock_P2;                  // Unlock p2
    p++;
    i++;
  }
  pthread_exit(NULL);
}
```

停止箇所(79)

```
void add_short_end(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    for ( ; ; )
    {
      Lock_P2;                      // Lock p2
      if ( TryLock_P1 == EBUSY )    // Lock p1
      {
        usleep(1);
        Unlock_P2;
        pw = p1, p1 = p2, p2 = pw;
//      printf("Busy:End (%d)¥n",i);
        continue;
      }
      break;
    }
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_P2;                  // Unlock p2
    Unlock_P1;                  // Unlock p1
    p--;
    i--;
  }
  pthread_exit(NULL);
}
```

停止箇所(117)

ライブロックしているとは
言えない

ライブロックの確認

- gdbによる確認(Ctrl-Cにより停止箇所を複数回確認)

```
void add_short_top(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = 0, p = array ; i < num ; )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    for ( ; ; )
    {
      Lock_P1;                      // Lock p1
      if ( TryLock_P2 == EBUSY )    // Lock p2
      {
        usleep(1);
        Unlock_P1;
        pw = p1, p1 = p2, p2 = pw;
//      printf("Busy:Top (%d)¥n",i);
        continue;
      }
      break;
    }
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_P1;                  // Unlock p1
    Unlock_P2;                  // Unlock p2
    p++;
    i++;
  }
  pthread_exit(NULL);
}
```

停止箇所(79)

```
void add_short_end(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    for ( ; ; )
    {
      Lock_P2;                      // Lock p2
      if ( TryLock_P1 == EBUSY )    // Lock p1
      {
        usleep(1);
        Unlock_P2;
        pw = p1, p1 = p2, p2 = pw;
//      printf("Busy:End (%d)¥n",i);
        continue;
      }
      break;
    }
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
                                  // Unlock p2
                                  // Unlock p1
```

停止箇所(117)

ライブロックして無かった

少しずつｉの値が変化する

ライブロックの確認

- gdbによる確認(ブレークポイント設定によりロック状態を確認)

```
void add_short_top(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = 0, p = array ; i < num ; )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    for ( ; ; )
    {
      Lock_P1;                      // Lock p1
      if ( TryLock_P2 == EBUSY )  // Lock p2
      {
        usleep(1);
        Unlock_P1;
        pw = p1, p1 = p2, p2 = pw;
//      printf("Busy:Top (%d)¥n",i);
        continue;
      }
      break;
    }
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_P1;                  // Unlock p1
    Unlock_P2;                  // Unlock p2
    p++;
    i++;
  }
  pthread_exit(NULL);
}
```

```
void add_short_end(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    for ( ; ; )
    {
      Lock_P2;                      // Lock p2
      if ( TryLock_P1 == EBUSY )  // Lock p1
      {
//      usleep(1);
        Unlock_P2;
        pw = p1, p1 = p2, p2 = pw;
        printf("Busy:End (%d)¥n",i);
        continue;
      }
      break;
    }
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_P2;                  // Unlock p2
    Unlock_P1;                  // Unlock p1
    p--;
    i--;
  }
  pthread_exit(NULL);
}
```

ブレークポイント(81)

iの値に変化なし

ライブロックを起こせたプログラム

- gdbによる確認

```
void add_short_top(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = 0, p = array ; i < num ; )
  {
    if ( i >= now_end )
      break;
    now_top=i;
    l = *p & 0xFFFF;
    u = (*p >> 16) & 0xFFFF;
    for ( ; ; )
    {
      Lock_P1;                      // Lock
      if ( TryLock_P2 == EBUSY )    // Lock
      {
        usleep(1);
        Unlock_P1;
        pw = p1, p1 = p2, p2 = pw;
//      printf("Busy:Top (%d)¥n",i);
        continue;
      }
      break;
    }
    short_sum_l += l;
    short_sum_u += u;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_P1;                  // Unlock p1
    Unlock_P2;                  // Unlock p2
    p++;
    i++;
  }
  pthread_exit(NULL);
}
```

```
void add_short_end(int *nump)
{
  int num=*nump,i,*p;
  unsigned short u,l;
  pthread_mutex_t *p1=&mutex_l,*p2=&mutex_u,*pw;

  for( i = num-1, p = array+(num-1) ; i >= 0 ; )
  {
    if ( i <= now_top )
      break;
    now_end=i;
    u = (*p >> 16) & 0xFFFF;
    l = *p & 0xFFFF;
    for ( ; ; )
    {
      Lock_P2;                      // Lock p2
      if ( TryLock_P1 == EBUSY )    // Lock p1
      {
//      usleep(1);
        Unlock_P2;
        pw = p1, p1 = p2, p2 = pw;
//      printf("Busy:End (%d)¥n",i);
        continue;
      }
      break;
    }
    short_sum_u += u;
    short_sum_l += l;
    if (short_sum_l < l)
      short_sum_u++;
    Unlock_P2;                  // Unlock p2
    Unlock_P1;                  // Unlock p1
    p--;
    i--;
  }
  pthread_exit(NULL);
}
```

この区間で
回避行動を
繰り返す

ライブロックは難しい

```
    for ( ; ; )
    {
      Lock_P1;                      // Lock p1
      if (TryLock_P2 == EBUSY)      // Lock p2
      {
        usleep(1);
        Unlock_P1;
        pw = p1, p1 = p2, p2 = pw;
//      printf("Busy:Top (%d)¥n",i);
        continue;
      }
      break;
    }
```

```
    for ( ; ; )
    {
      Lock_P2;                      // Lock p2
      if (TryLock_P1 == EBUSY)      // Lock p1
      {
//      usleep(1);
        Unlock_P2;
        pw = p1, p1 = p2, p2 = pw;
//      printf("Busy:End (%d)¥n",i);
        continue;
      }
      break;
    }
```

- ライブロックでは、互いが相手のスレッドに対してミューテックス変数ロックの譲歩を繰り返すループ状況が起きる
- しかし、完全に処理が進まないわけではなく、何かの拍子に少し動くことがある
- 結果的に、全体の処理がおそろしく遅くなってしまう
- では、なぜ時々動くのか、何の拍子で動くのか

プログラムコードは見た目ほど単純ではない

- **関数呼び出しにはそれなりの時間かかかる**
  引数設定・コール・関数の前処理・関数内の処理・戻り値の設定・関数の後処理・リターン
- **動いているのは2つのスレッドだけではない**
  多くのプロセスが限られたCPUコアを争って使うために、たびたび待たされる
- **ループの1回転はいつも同じ時間では動かない**
  数行のプログラムであっても、それなりのコードサイズがあり、スレッドは不定期に停止しながら動作している
- **少しずつのズレが、BUSY状態から抜ける条件をまれに満たす**

```
    for ( ; ; )
    {
      Lock_P1;                          // Lock p1
      if ( TryLock_P2 == EBUSY )        // Lock p2
      {
        usleep(1);
        Unlock_P1;
        pw = p1, p1 = p2, p2 = pw;
//      printf("Busy:Top (%d)¥n",i);
        continue;
      }
      break;
    }
```

printf()を有効にすると
可変引数処理を含む書式の解析、数値変換、
文字列生成、標準出力という膨大な処理が
動く

ブログは継続中

■ 情報は随時公開中

https://www.embeddedmulticore.org/ にて

「マルチコア適用ガイド」V1.0 公開

■ EMCホームページにて

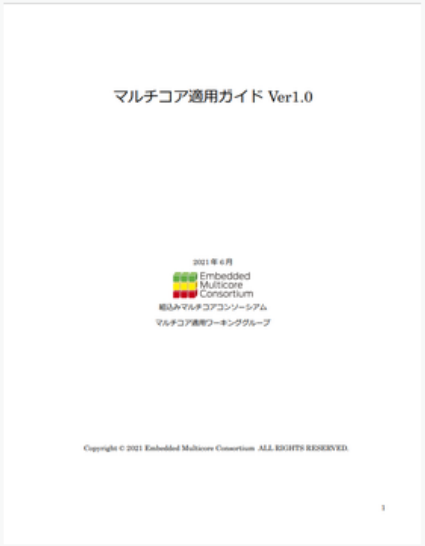
NEWS · 2021/09/30

2020 EMC「マルチコア適用ガイド」資料公開

「マルチコア適用ガイド」統合版にてリニュアル致しました。下記「ダウンロード」ボタンより資料ご参照ください。また、ガイドに関して、下記のコメント欄から皆さまのご意見をお聞かせ下さい。

マルチコア適用ガイドVer1.0.pdf
PDFファイル [10.2 MB]

ダウンロード



<"マルチコア適用ガイド"の各章>

1章 <並列化フロー>完成度の高いマルチコアソフトウェアを効率よく作成するための開発手順
2章 <動作の見える化>マルチコアの問題解決に役立つ可視化の技術
3章 <テスト設計>マルチコア用プログラムを対象としたテストの勘所
4章 <品質評価>組込みシステムをマルチコア化したときに確保すべき品質とは
5章 <自動車応用>車載システム向けのドメインごとの特徴とマルチコア対応
6章 制御系マルチコア・ハードウェアの特徴とユースケース
7章 自動車 機能安全へのマルチコア適用
8章 並列処理ソフトウェアの課題と対策技術
9章 <Appendix>組込みマルチコア用語集

★お願い★
「マルチコア適用ガイド」につきまして、ご意見をもとにより使いやすいガイドに改良・改版してまいります。皆様のご意見をお待ちしております。下記のダウンロードのページ（Blog内）からご記入下さい。

Embedded
Multicore
Consortium

www.embeddedmulticore.org

お問い合わせは

www.embeddedmulticore.org

Embedded
Multicore
Consortium