z/TPF V1.1

TPF Users Group Fall 2007
Title: Threads Support in z/TPF

Name: Mark Cooper
Venue: Applications Subcommittee

AIM Enterprise Platform Software
IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Project Description – PJ31976/PJ31977

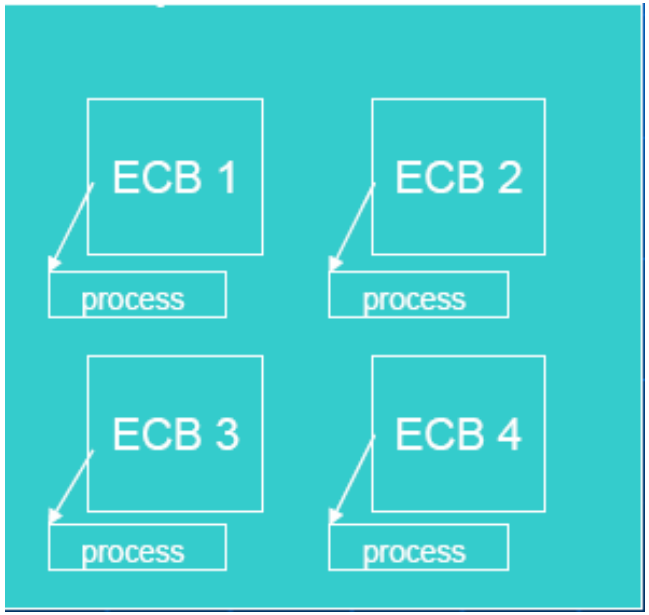
1. Enable z/TPF to run POSIX threads (pthreads) Support Majority of the standard
2. IEEE POSIX 1003.1c standard pthread APIs(www.opengroup.org)

pthreads on z/TPF

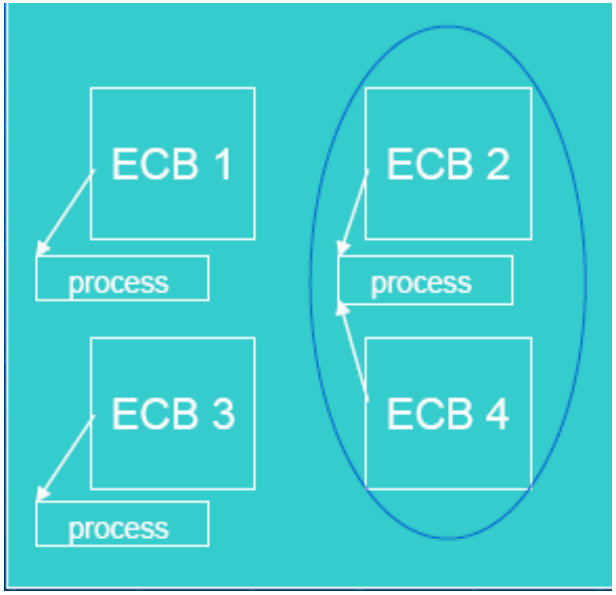
- A thread on TPF roughly corresponds to an ECB, where a group of thread ECBs are considered part of a process.
- A thread process is created when a running ECB issues a thread API, such as the pthread_create function. That ECB is called the master thread ECB.
- All of the threads that originate from the master thread ECB, or thread ECBs in the process, are part of the process.
- When the master thread ECB exits, all thread ECBs in the process exit.
- No thread has higher priority (unless specified) than any other thread in the process. Thread priority does not affect TPF APIs.
- Threads have existed on TPF since TPF 4.1. What's new? API now available for general use; thread-safe libraries/services.

TPF pthreads

TPF system without threads



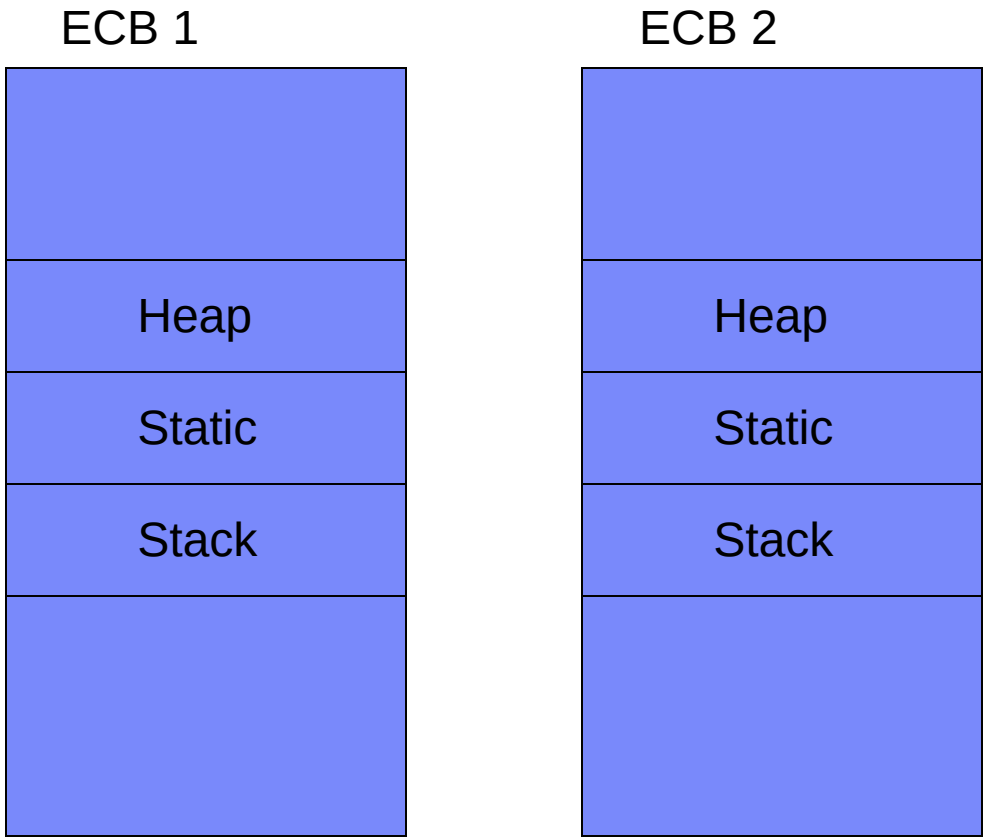
TPF system with threads



Thread ECBs within a process…

- **Run concurrently**
- **Share a single address space**
  - ECB heap, application stack, and static data
  - Not TPF unique blocks

Address Space – 2 ECBs

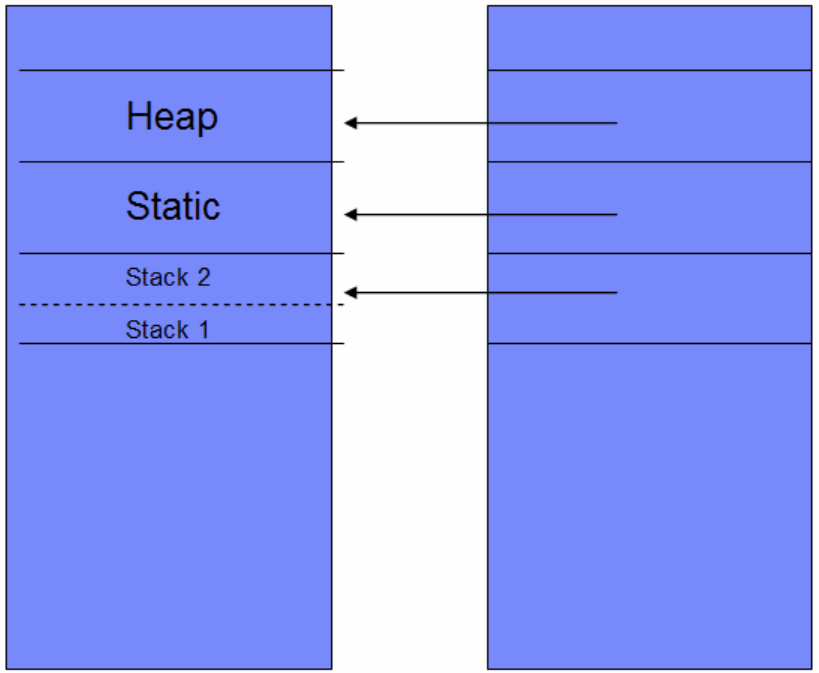


Address Space – 2 Threads

ECB THREAD 1

(Master thread ECB)

ECB THREAD 2

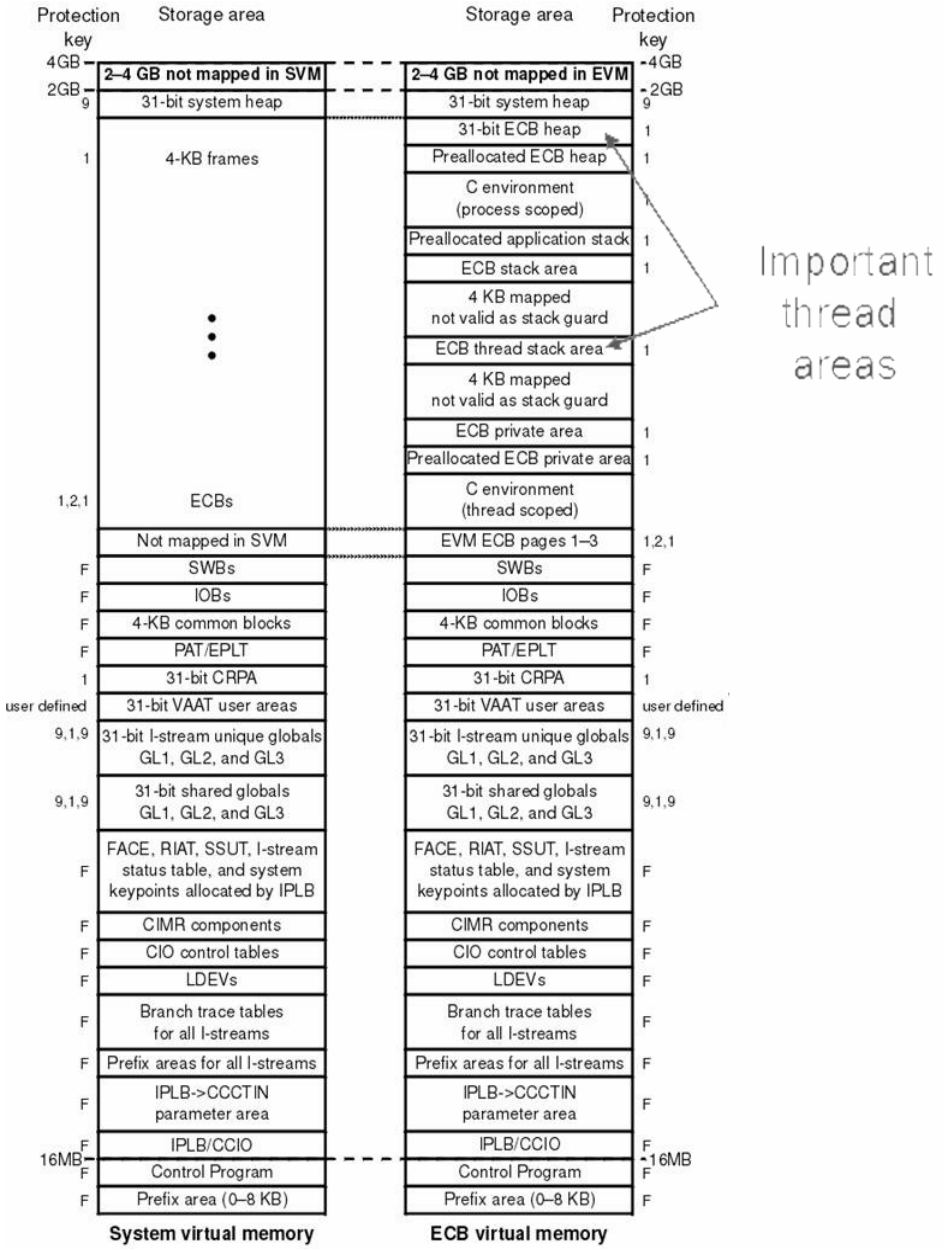


Thread Storage

- As each thread is created in a process, the ECB heap area of the master thread is shared with the newly created thread. Because there are more ECBs sharing the same amount of ECB heap storage, you might need to increase the amount of ECB heap storage.

- When a thread issues a request for ECB heap storage, after the pre-allocated heap area is exhausted, 1-MB frames are attached to the master thread. These frames are not released until the master thread exits.

Virtual Storage Layout



Layout of both ECB and system virtual memory

ZCTKA

- To enable threads, enter the ZCTKA ALTER command and specify the MTHD and TSTK parameters.
- The sum of the application stack for a thread entry control block (ECB), the application stack for an ECB, and the 31-bit ECB heap must be less than 2 GB.

Three ways to create a new ECB

1. TPF unique functions (cremc, …) - ECB
2. fork() - Process
3. pthread_create() - Thread

TPF unique functions (cremc, …) - ECB

- Does nothing with the ISO-C environment (ECB heap (31-bit heap and 64-bit heap); application stack; and static data) from the calling process.
- Part of the traditional TPF environment is passed (ECB work area).
- Need to create a separate program because execution will start at the top of the new program (or can play games with existing program).
- Quickest way to create a new ECB.
- Can use SystemV semaphores and signals (kill()) between ECBs.
- Any intercommunication between the caller and created ECB is designed and implemented by the user.

fork() - Process

- Copies all of the ISO-C environment (ECB heap (31-bit heap and 64-bit heap); application stack; and static data) from the calling process into the new process (including open file descriptors). In addition, if the process is threaded, the thread environment will be copied.
- Traditional TPF environment is not copied.
- Both processes will resume execution at the same location in the program following the fork(). Easy to pass data to the new ECB.
- Don't have to create a separate program
- Not the quickest way to create a new process.
- Can use SystemV semaphores and signals (kill()) between processes.
- Parent notified when the child exits (SIGCHLD signal).

pthread_create() - Thread

- Shares all of the ISO-C environment (ECB heap (31-bit heap and 64-bit heap); application stack; and static data) from the calling thread and the new thread (including open file descriptors).
- Traditional TPF environment is not copied or shared.
- Easier to pass data to the new ECB than TPF unique functions that create ECBs, but not as easy fork().
- Don't have to create a separate program but need a separate function.
- There is some additional overhead creating a threaded process. Each thread created is similar to creating an ECB.
- Can use Posix semaphores between threads.

Strategic Advantages

- Heavily used in the industry
- Application productivity improvement
- Open system standard APIs

Uses for threads on TPF

1. Ported code containing pthread calls
2. New code written by non-TPF application programmers (ie. college grads)
3. New code written where data needs to be shared.

z/TPF supported APIs

- Majority of the POSIX standard pthread
- functions and z/TPF-unique thread functions

Supported pthread APIs

1.Basic functions

i)Basic thread management

a.pthread_create
b.pthread_exit
c.pthread_join
d.pthread_once
e.pthread_kill
f.pthread_self
g.pthread_equal
h.pthread_yield (not part of the standard)
i.pthread_detach
j.pthread_delay_np (non-portable and not part of the standard)

ii)Thread-specific data

a.pthread_key_create
b.pthread_key_delete
c.pthread_getspecific
d.pthread_setspecific

iii)Thread cancellation

a.pthread_cancel (an extension of the standard)
b.pthread_cleanup_pop
c.pthread_cleanup_push
d.pthread_setcancelstate
e.pthread_testcancel

iv)Signals

a.pthread_sigmask
b.sigwait (in signal.h)

Supported pthread APIs cont.

2.Pthread Attribute Functions

i)Basic management

a.pthread_attr_init

b.pthread_attr_destroy

ii)Detachable or joinable

a.pthread_attr_setdetachstate

b.pthread_attr_getdetachstate

iv)Thread scheduling attributes

a.pthread_attr_getschedparam

b.pthread_attr_setschedparam

c.pthread_attr_getschedpolicy

d.pthread_attr_setschedpolicy

e.pthread_attr_setinheritsched

f.pthread_attr_getinheritsched

g.pthread_getschedparam (not part of the standard)

h.pthread_setschedparam (not part of the standard)

Supported pthread APIs cont.

3.Mutex Functions
- i)Basic mutex management
  - a.pthread_mutex_init
  - b.pthread_mutex_destroy
  - c.pthread_mutex_lock
  - d.pthread_mutex_unlock
  - e.pthread_mutex_trylock

4.Mutex Attribute Functions
- i)Basic mutex management
  - a.pthread_mutexattr_init
  - b.pthread_mutexattr_destroy
  - c.pthread_mutexattr_settype (an extension of the standard)
  - d.pthread_mutexattr_gettype (an extension of the standard)

5.Condition Variable Functions
- i)Basic condition variable management
  - a.pthread_cond_init
  - b.pthread_cond_destroy
  - c.pthread_cond_signal
  - d.pthread_cond_broadcast
  - e.pthread_cond_wait
  - f.pthread_cond_timedwait
  - g.pthread_get_expiration_np (non-portable and not part of the standard)

6.Condition variable Attribute Functions
- i)Basic mutex management
  - a.pthread_condattr_init
  - b.pthread_condattr_destroy

Supported pthread APIs cont.

7.Read/write locks Functions (all an extension of the standard)

i)Basic read/write lock management

a.pthread_rwlock_init

b.pthread_rwlock_destroy

c.pthread_rwlock_rdlock

d.pthread_rwlock_tryrdlock

e.pthread_rwlock_trywrlock

f.pthread_rwlock_unlock

g.pthread_rwlock_wrlock

8.Read/write locks Attribute Functions (all an extension of the standard)

i)Basic read/write lock management

a.pthread_rwlockattr_init

b.pthread_rwlockattr_destroy

Supported pthread APIs cont.

9. Macros
   a. PTHREAD_CANCEL_ENABLE
   b. PTHREAD_CANCEL_DISABLE
   c. PTHREAD_CANCELED
   d. PTHREAD_COND_INITIALIZER
   e. PTHREAD_CREATE_DETACHED
   f. PTHREAD_CREATE_JOINABLE
   g. PTHREAD_EXPLICIT_SCHED
   h. PTHREAD_INHERIT_SCHED
   i. PTHREAD_MUTEX_DEFAULT (an extension of the standard)
   j. PTHREAD_MUTEX_ERRORCHECK (an extension of the standard)
   k. PTHREAD_MUTEX_INITIALIZER
   l. PTHREAD_MUTEX_NORMAL (an extension of the standard)
   m. PTHREAD_MUTEX_RECURSIVE (an extension of the standard)
   n. PTHREAD_ONCE_INIT

Supported pthread APIs cont.

10. POSIX semaphore APIs
    a. sem_init
    b. sem_destroy
    c. sem_wait
    d. sem_trywait
    e. sem_post
    f. sem_getvalue
    g. sem_timedwait

Supported pthread APIs cont.

11. TPF unique API for threads

- tpf_pthread_options() - Determines the behavior of threads within a process when this thread exits unexpectedly (that is, in any way other than by using the pthread_exit function or returning).

  Default: The multithreaded process exits, the standard POSIX behavior.

  Note: Not inherited by threads.

Posix C APIs that operate on a thread level

1. sleep()
2. sigsuspend()
3. sigprocmask()
4. sigpending()
5. raise()
6. pause()

Data reduction

```
THREADS SUPPORT DATA                                                                    MIN
   MAX     MEAN
   MAXIMUM THREADS PER PROCESS                                                  0.000    26.000
   23.440


   HIGH WATER MARK NUMBER OF THREADS ACTIVE IN A PROCESS: 26


   THREAD CREATION RATE:                                                                12.489


   THREADED PROCESSES CREATED DURING DATA COLLECTION       362


THREADED PROCESS SUMMARY REPORT (THREADS PER PROCESS PER SECOND)
    PID             NAME               MIN                 MAX                 MEAN
==========    ====   =========   ==========   ===========
1074200709    CSL2                0.00                0.00                0.00
1075773525    CSL2                0.00                0.00                0.00
OTHER                                                                                0.03
```

Thread dump info

TPF unique thread number within the process

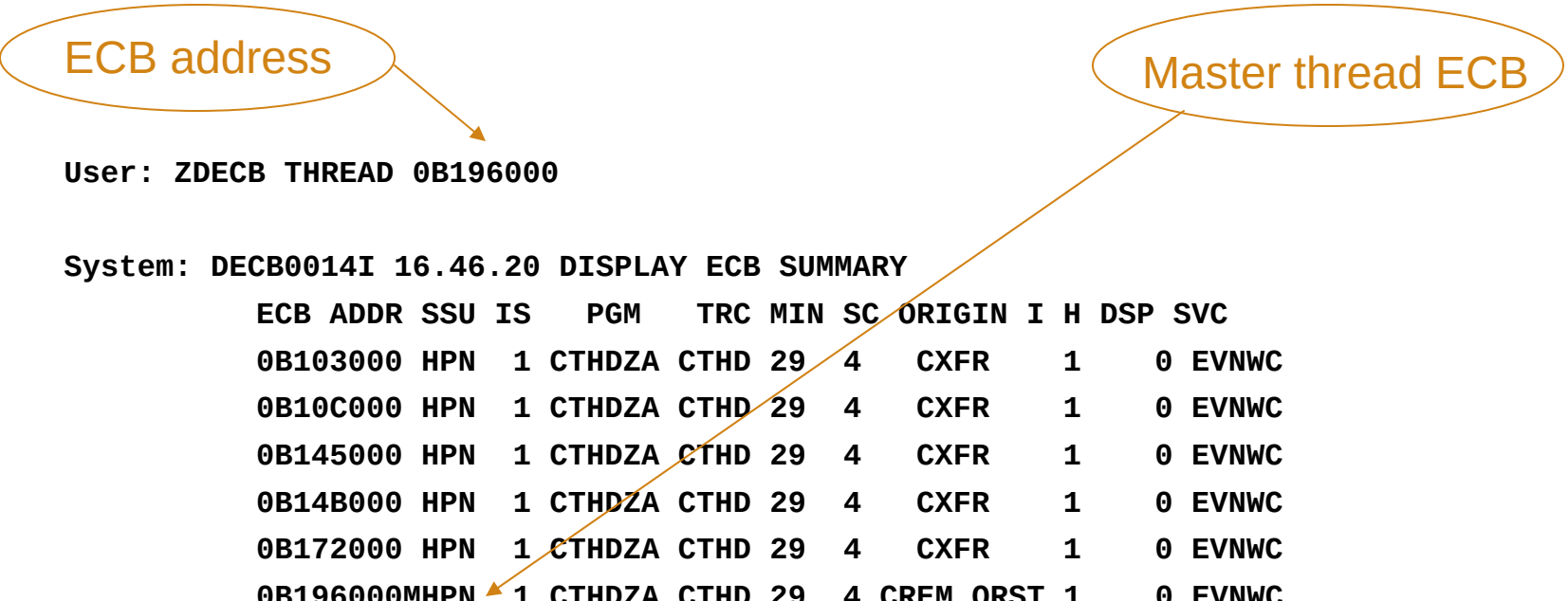
ENTRY REFERENCED VIA R9 - DUMP OF ECB VIRTUAL MEMORY FOLLOWS FOR THREAD

PROCESS ID: 401F000B
THREAD ID: 00000001

***ENTRY BLK, GENERATED BY CNTRL-XFER**

000000000AB00000 CHW 00000000 BAD 00000000 W00 C4C5C3C2 010000C2
000000000AB00010 08 80B00000 00000001 016 00000000 4BE3C940
000000000AB00020 024 00090106 00510FFF 032 CF000C00 00000192
000000000AB00030 040 0100000E 0265E3B0 048 C3D7E4C2 E2D4D7C2

ZDECB




```
User: ZDECB THREAD 0B196000

System: DECB0014I 16.46.20 DISPLAY ECB SUMMARY
          ECB ADDR SSU IS   PGM   TRC MIN SC ORIGIN I H DSP SVC
          0B103000 HPN  1 CTHDZA CTHD 29  4   CXFR    1    0 EVNWC
          0B10C000 HPN  1 CTHDZA CTHD 29  4   CXFR    1    0 EVNWC
          0B145000 HPN  1 CTHDZA CTHD 29  4   CXFR    1    0 EVNWC
          0B14B000 HPN  1 CTHDZA CTHD 29  4   CXFR    1    0 EVNWC
          0B172000 HPN  1 CTHDZA CTHD 29  4   CXFR    1    0 EVNWC
          0B196000MHPN  1 CTHDZA CTHD 29  4 CREM QRST 1    0 EVNWC
          0B1A5000 HPN  1 CTHDZA CTHD 29  4   CXFR    1    0 EVNWC
          0B1BA000 HPN  1 CTHDZA CTHD 29  4   CXFR    1    0 EVNWC
          0B1D8000 HPN  1 CTHDZA CTHD 29  4   CXFR    1    0 EVNWC
          0B220000 HPN  1 CTHDZA CTHD 29  4   CXFR    1    0 EVNWC
          TOTAL 10
          END OF DISPLAY
```

ZSTAT

New created stat for threads

```
User: ZSTAT

System: STAT0014I 22.46.00 SYSTEM STATUS DISPLAY _
                            IOB FRAME COMMON  SWB ECB FRM1MB
        ALLOCATED          2704  5000    250 1252 300    300
        AVAILABLE          2704  4973    247 1210 294    189 _

        ACTIVE ECBS           6
        DLY/DFR ECBS          0
        PROCESSED           652
        LOW SPEED             0
        ROUTED                0
        CREATED           77637
        THREADS CREATED     188
        SNA                   0 _
        TCP/IP INPUT          0
        TCP/IP OUTPUT         0
        END OF DISPLAY
```

ZASER

```
User:     ZASER THEAPON

System: ASER0000I 14.12.04 - OK+
```

Includes the heap storage of the entire process if the ECB is running in a threaded environment.

Signals

- Two ways to send signals: kill() sends a signal to a process; pthread_kill() sends a signal to a thread.
- A signal cannot be directed to a specific thread if kill() is used.
- It’s preferable to use APIs specifcally designed for threads (ie., pthread_kill(), pthread_sigmask(), sigwait())

Thread Scheduling

Thread scheduling APIs affect only thread queues, such as a thread that is waiting to acquire a mutex lock. z/TPF system services use standard z/TPF scheduling.

Thread-safe System services

- z/TPF services are thread-safe (protect shared resources from being accessed concurrently).
- z/TPF libraries and system services contain thread-safe APIs unless noted otherwise.

Thread Inheritance

Relevant z/TPF information is copied from the caller of pthread_create to a new thread. Copied information includes, but is not limited to:

- Cycle down indicators
- Database ID
- ECB activation number
- ECB owner name
- ECB resource monitor indicators
- Functional CRAS support indicators
- PAT entry address for the active program
- Program base ID
- SSU ID
- Time slice information
- Trace control transfer/creates.

Semaphores

POSIX semaphores on z/TPF work only between threads in a process, not between processes (the sem_init function has a restricted value for the __**pshared** parameter; the __**pshared** parameter can only be set to 0).

Note: To use semaphores between processes, use TPF's SystemV semaphore support.

Example

```
sem_t produced, consumed;
int n;
```

```
void *produce(void *arg) {
    int i, loopcnt;
    loopcnt = (int)(long)arg;
    for (i=1; i<=loopcnt; i++)    {
        sem_wait(&consumed);
        n++;
        sem_post(&produced);
        }
    return 0;
}
```

```
void *consume(void *arg) {
    int i, loopcnt;
    loopcnt = (int)(long)arg;
    for (i=1; i<=loopcnt; i++)    {
        sem_wait(&produced);
        printf("n is %d\n", n);
        sem_post(&consumed);
        }
    return 0;
}
```

```
int SCENARIO25() {
  pthread_t idprod, idcons; /* ids of threads */
  long loopcnt = 5;

   n = 1;
   sem_init(&consumed, 0, 0);
   sem_init(&produced, 0, 1);
   pthread_create(&idprod, NULL, produce, (void *)loopcnt);
   pthread_create(&idcons, NULL, consume, (void *)loopcnt);
   pthread_join(idprod, NULL);
   pthread_join(idcons, NULL);
   sem_destroy(&produced);
   sem_destroy(&consumed);
   return 0;
}
```

Trademarks

- Update the following as appropriate. refer to http://www.ibm.com/legal/copytrade.shtml and don't leave this line in
- IBM, xxx and xxxx are trademarks of International Business Machines Corporation in the United States, other countries, or both.
- delete any of the following which are not used. Update the Windows trademark to reflect only those items used. delete this line
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Intel, Intel Inside (logos), MMX, Celeron, Intel Centrino, Intel Xeon, Itanium, Pentium and Pentium III Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, or service names may be trademarks or service marks of others.
- Notes
- Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
- All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
- This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
- All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
- Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
- Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.
- This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.