ｆｏｒｋのトレース

mm（メモリ管理）で do_fork を探すと、ファイル forkexit.c に do_fork があることがわかります。

/usr/src/kernel/mm/forkexit.c の先頭のコメントを見てみると、以下のように書かれています。（英語なので、「説明」であることに気づかない人もいるかも知れませんが・・・）

```
/* This file deals with creating processes (via FORK) and deleting them (via
 * EXIT/WAIT).  When a process forks, a new slot in the 'mproc' table is
 * allocated for it, and a copy of the parent's core image is made for the
 * child.  Then the kernel and file system are informed.  A process is removed
 * from the 'mproc' table when two events have occurred: (1) it has exited or
 * been killed by a signal, and (2) the parent has done a WAIT.  If the process
 * exits first, it continues to occupy a slot until the parent does a WAIT.
 *
 * The entry points into this file are:
 *   do_fork:    perform the FORK system call
 *   do_mm_exit: perform the EXIT system call (by calling mm_exit())
 *   mm_exit:    actually do the exiting
 *   do_wait:    perform the WAITPID or WAIT system call
 */
```

ここで、do_fork を調べてみます。

```
PRIVATE pid_t next_pid = INIT_PID+1;    /* next pid to be assigned */

FORWARD _PROTOTYPE (void cleanup, (register struct mproc *child) );

/*===========================================================================*
 *                                do_fork                                    *
 *===========================================================================*/
PUBLIC int do_fork()
{
/* The process pointed to by 'mp' has forked.  Create a child process. */

  register struct mproc *rmp;   /* pointer to parent */
  register struct mproc *rmc;   /* pointer to child */
  int i, child_nr, t;
  phys_clicks prog_clicks, child_base;
  phys_bytes prog_bytes, parent_abs, child_abs; /* Intel only */

 /* If tables might fill up during FORK, don't even start since recovery half
  * way through is such a nuisance.
  */
  rmp = mp;
  if (procs_in_use == NR_PROCS) return(EAGAIN);
  if (procs_in_use >= NR_PROCS-LAST_FEW && rmp->mp_effuid != 0)return(EAGAIN);
```

ＭＩＮＩＸでは、最大に起動できるプロセス数に制約があるようです。

変数 proc_in_use が NR_PROCS と比較されています。

```
# grep NR_PROCS /usr/include/minix/*.h
/usr/include/minix/config.h:#define NR_PROCS          32
```

この値がいくつか調べてみると、３２であることがわかります。

MINIX は、学習用の OS ですから、これ以上必要ないということです。

どのくらいメモリを割り付けるか決める、という作業が最初にあります。

```
  /* Determine how much memory to allocate.  Only the data and stack need to
   * be copied, because the text segment is either shared or of zero length.
   */
  prog_clicks = (phys_clicks) rmp->mp_seg[S].mem_len;
  prog_clicks += (rmp->mp_seg[S].mem_vir - rmp->mp_seg[D].mem_vir);
  prog_bytes = (phys_bytes) prog_clicks << CLICK_SHIFT;
  if ( (child_base = alloc_mem(prog_clicks)) == NO_MEM) return(ENOMEM);

  /* Create a copy of the parent's core image for the child. */
  child_abs = (phys_bytes) child_base << CLICK_SHIFT;
  parent_abs = (phys_bytes) rmp->mp_seg[D].mem_phys << CLICK_SHIFT;
  i = sys_copy(ABS, 0, parent_abs, ABS, 0, child_abs, prog_bytes);
  if (i < 0) panic("do_fork can't copy", i);

  /* Find a slot in 'mproc' for the child process.  A slot must exist. */
  for (rmc = &mproc[0]; rmc < &mproc[NR_PROCS]; rmc++)
        if ( (rmc->mp_flags & IN_USE) == 0) break;

  /* Set up the child and its memory map; copy its 'mproc' slot from parent. */
  child_nr = (int)(rmc - mproc);        /* slot number of the child */
  procs_in_use++;
  *rmc = *rmp;                  /* copy parent's process slot to child's */
```

<< CLICK_SHIFT

という演算が頻繁に使われていることがわかりますか？<< は、C 言語ばかりではなく Java でも共通ですが、左シフト演算です。

２進数を左に１回シフトすると２倍されます。

```
# grep CLICK_SHIFT /usr/include/minix/*.h
/usr/include/minix/const.h:#define CLICK_SHIFT       10 /* log2 of CLICK_SIZE */
/usr/include/minix/const.h:#define CLICK_SHIFT    12     /* 2log of CLICK_SIZE */
```

CLICK_SHIFT は２箇所定義されていますが、（２箇所ということはありません、スイッチによってどちらかに切り替えられているはずですが、）

```
/* Memory is allocated in clicks. */
#if (CHIP == INTEL)
#define CLICK_SIZE      1024    /* unit in which memory is allocated */
#define CLICK_SHIFT       10    /* log2 of CLICK_SIZE */
#endif

#if (CHIP == SPARC) || (CHIP == M68000)
#define CLICK_SIZE      4096    /* unit in which memory is alocated */
#define CLICK_SHIFT       12    /* 2log of CLICK_SIZE */
#endif
```

この記述を見て、皆さんのコンピュータで、CLICK_SHIFT がいくつに定義されているか、わかるでしょうか？また、メモリを管理する単位が、何バイトかわかりますか？

次に、child_abs や parent_abs という変数名ですが、変数名からこれらがどんな意味を持っているか想像できますか？

これらの変数が使われている次の行、sys_copy は、/usr/src/lib/syslib で定義

されています。コメントに、全ての答えが書いてあります。

```
int dst_proc;                   /* dest process */
int dst_seg;                    /* dest segment: T, D, or S */
phys_bytes dst_vir;             /* dest virtual address (phys addr for ABS) */
phys_bytes bytes;               /* how many bytes */
{
/* Transfer a block of data.  The source and destination can each either be a
 * process (including MM) or absolute memory, indicate by setting 'src_proc'
 * or 'dst_proc' to ABS.
 */

  message copy_mess;

  if (bytes == 0L) return(OK);
  copy_mess.SRC_SPACE = src_seg;
  copy_mess.SRC_PROC_NR = src_proc;
  copy_mess.SRC_BUFFER = (long) src_vir;

  copy_mess.DST_SPACE = dst_seg;
  copy_mess.DST_PROC_NR = dst_proc;
  copy_mess.DST_BUFFER = (long) dst_vir;

  copy_mess.COPY_BYTES = (long) bytes;
  return(_taskcall(SYSTASK, SYS_COPY, &copy_mess));
}
```

コピーの後で、71 行目、procs_in_use という変数がインクリメントされています。この変数の意味を調べることも、大切です。(名称から、意味はある程度想像できますが・・・)

77 行目からは、セグメントに関する処理を行っています。
セグメントに関する格納域は三つあります。それぞれ、T（テキスト）と D（データ）と S（スタック）ですが、「こうでなければならない」ということが書かれています。英語のコメントを読んでください。

これら、一連の処理が終わると、fs や kernel に「プロセスの fork が行われた」ことを通知します。

```
  /* Tell kernel and file system about the (now successful) FORK. */
  sys_fork(who, child_nr, rmc->mp_pid);
  tell_fs(FORK, who, child_nr, rmc->mp_pid);
```

以上が、メモリ管理についての、トレースのヒントです。
（あまり、ヒントになっていない？）

レポートは、プログラムの流れに沿って
『_____という処理を実行すると、_____というファイルの____行目で、以下のような
　　_____（プログラム行の引用）_____
　記述がある。これは、_____を_____処理している。

　つまり「＿＿＿＿＿（セグメント、物理メモリなどのキーワード)」は、＿＿＿＿＿
　処理されていることがわかる。
』
という形式で書かれていると、内容の理解度が伝わりやすいと思います。

英語のコメントを読むだけで、かなり動作の説明になりますが、レポートは一応英語ではなく日本語で書いて下さい。
英語でレポートを書くならば、最初から最後まで一貫して英語で書いてある場合には、それなりの評価を行います。