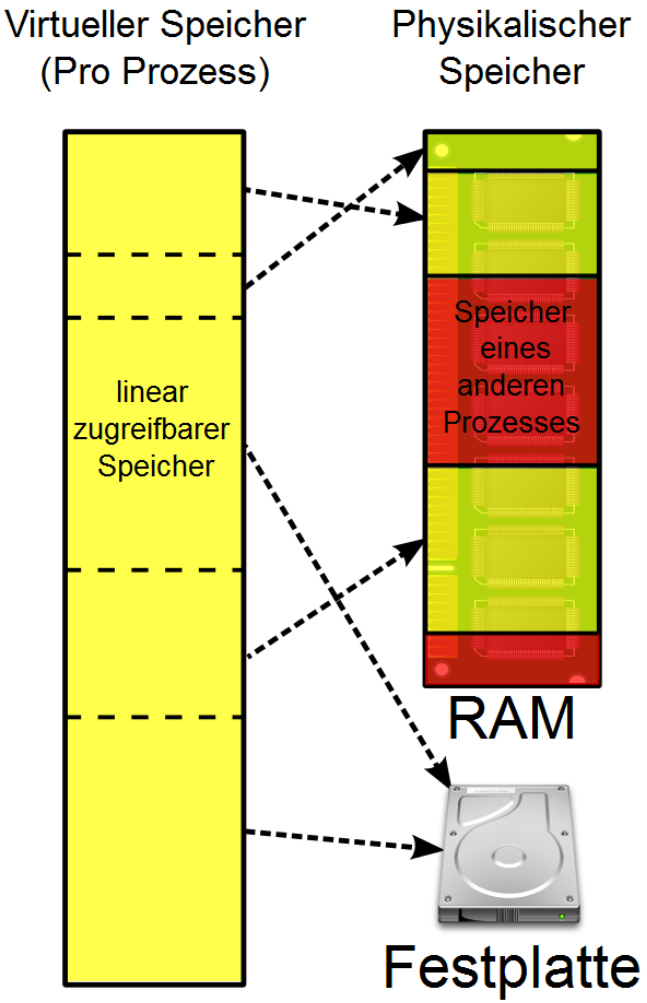
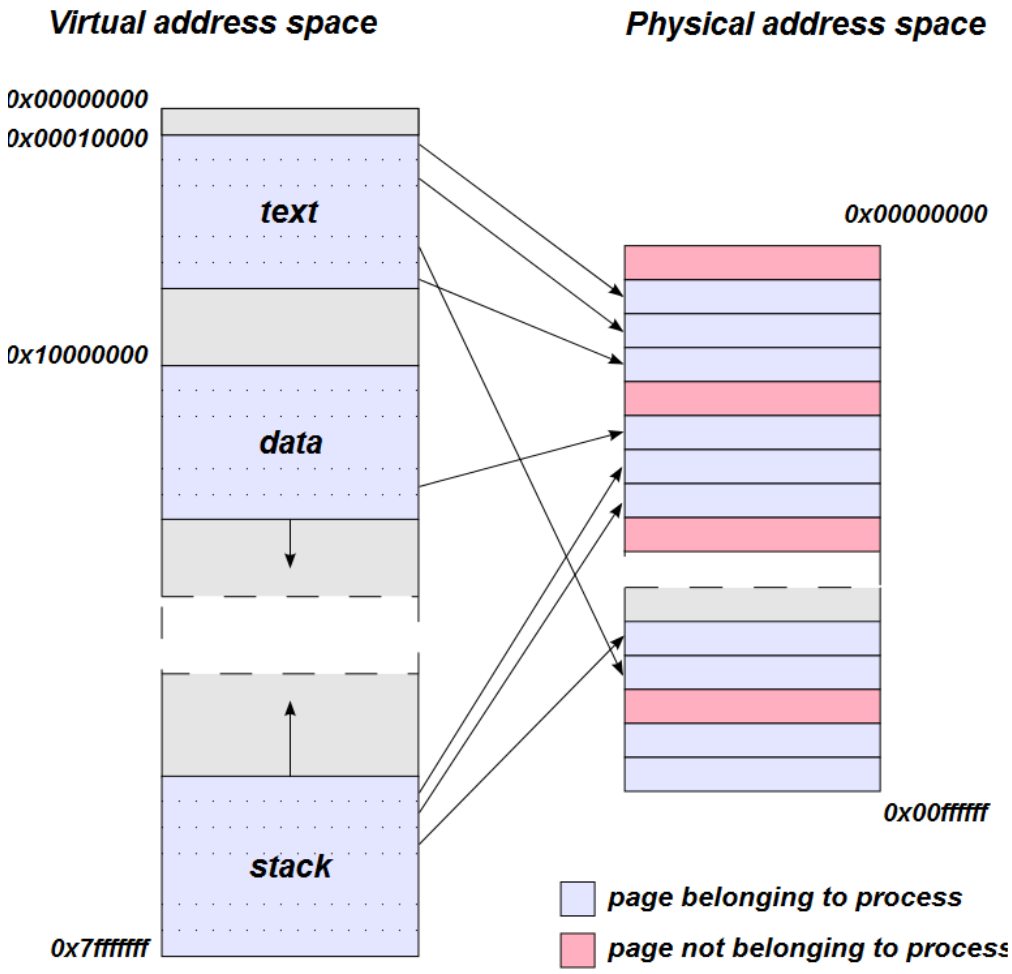
Betriebssystemressourcen mit C ansprechen

w.mueller@tsn.at 2020

Virtuelle Speicherverwaltung





Quelle: http://de.wikipedia.org/wiki/Virtuelle_Speicherverwaltung

Posix – Standardisierung

http://www.opengroup.org/unix/online.html

Portable Operating System Interface (POSIX) by The Open Group

Speicher anfordern mit calloc

http://manpages.ubuntu.com/manpages/precise/en/man3/calloc.3posix.html

NAME

calloc - a memory allocator

SYNOPSIS

#include <stdlib.h> void *calloc(size_t nelem**, size_t** elsize**);**

DESCRIPTION

The calloc() function shall allocate unused space for an array of nelem elements each of whose size in bytes is elsize. The space shall be initialized to all bits 0. The order and contiguity of storage allocated by successive calls to calloc() is unspecified.

....

RETURN VALUE

Upon successful completion with both nelem and elsize non-zero, calloc() shall return a pointer to the allocated space. If either nelem or elsize is 0, then either a null pointer or a unique pointer value that can be successfully passed to free() shall be returned. Otherwise, it shall return a null pointer and set errno to indicate the error.

Speicher frei geben mit free

http://manpages.ubuntu.com/manpages/precise/en/man3/free.3posix.html

NAME

free - free allocated memory

SYNOPSIS

#include <stdlib.h> void free(void *ptr);

DESCRIPTION

The free() function shall cause the space pointed to by ptr to be deallocated; that is, made available for further allocation. If ptr is a null pointer, no action shall occur. Otherwise, if the argument does not match a pointer earlier returned by the calloc(), malloc(), posix_memalign(), realloc(), or strdup() function, or if the space has been deallocated by a call to free() or realloc(), the behavior is undefined. Any use of a pointer that refers to freed space results in undefined behavior.

RETURN VALUE

The free() function shall not return a value.

Beispiel – calloc - free

```
#include <stdlib.h>

const size_t M=1024*1024; // one Megabyte
char *buffer, *p;

int main() {
  buffer=calloc(10, M); // allocate 10 Megabytes of memory
   p=buffer; int i ;
  for (i=0; i<M ; i++) *(p++)=0xff; // fill memory
  free(buffer);
}
```

CPU verbraten

```
// eine Möglichkeit:
#include <math.h>

void burn_cpu(int ntimes) {
    int j; double x=2;
    printf(
     "... cpu loop %d Mio times x=sin(x)\n",ntimes);
    for( j=0; j<ntimes; j++) {
         int i; for (i=0;i<M ; i++) x=sin(x);
    };
}
```

POSIX threads

- In einem Prozess kann ein Unterprogramm als unabhängiger thread gestartet werden.
- Der thread teilt sich die globalen Variablen und besitzt eigene lokale Variablen.
- Er bekommt eigene CPU Ressourcen und läuft parallel zu den anderen Threads des Programms.

POSIX threads

```
#include <pthread.h>

int cpu_loop_cnt=100;
void my_thread() { burn_cpu(cpu_loop_cnt); }

int main() {
    pthread_t thp[10] ;  /Feld für 10 thread Strukturen
    int i, n=3;
    for(i=0; i<n ; i++) { // threats anlegen und starten
      if ( pthread_create(&thp[i],NULL,(void*)&my_thread ,NULL)
)
          perror ( "could not create thread" );
    };
    for(i=0;i<n ; i++) {
       pthread_join ( thp[i] , NULL) ;  // auf Ende warten
    }
    printf(„Alles fertig\n“);
}
```

POSIX threads - pthread_create

http://manpages.ubuntu.com/manpages/lucid/en/man3/pthread_create.3posix.html

NAME

pthread_create - thread creation

SYNOPSIS

#include <pthread.h>

int pthread_create(pthread_t *restrict thread**, const pthread_attr_t *restrict** attr**, void *(*** start_routine**)(void*), void *restrict** arg**);**

DESCRIPTION

The pthread_create() function shall create a new thread, with attributes specified by attr, within a process. If attr is NULL, the default attributes shall be used. If the attributes specified by attr are modified later, the thread's attributes shall not be affected.
Upon successful completion, pthread_create() shall **store the ID of the created thread in the location referenced by thread**. The thread is created executing start_routine with arg as its sole argument.
If the start_routine returns, the effect shall be as if there was an implicit call to pthread_exit() using the return value of start_routine as the exit status. Note that the thread in which main() was originally invoked differs from this. When it returns from main(), the effect shall be as if there was an implicit call to exit() using the return value of main() as the exit status. ….

RETURN VALUE

If successful, the pthread_create() function shall return zero; otherwise, an error number shall be returned to indicate the error.

…..

POSIX threads - pthread_join

http://manpages.ubuntu.com/manpages/lucid/en/man3/pthread_join.3posix.html

NAME

pthread_join - wait for thread termination

SYNOPSIS

#include <pthread.h>

int pthread_join(pthread_t thread**, void ****value_ptr**);**

DESCRIPTION

The pthread_join() function shall suspend execution of the calling thread until the target thread terminates, unless the target thread has already terminated.
On return from a successful pthread_join() call with a non-NULL value_ptr argument, the value passed to pthread_exit() by the terminating thread shall be made available in the location referenced by value_ptr.
When a pthread_join() returns successfully, the target thread has been terminated.
…

RETURN VALUE

If successful, the pthread_join() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ganzes Programm (1/4)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include <time.h>
#include <pthread.h>
// http://stackoverflow.com/questions/19266786/how-do-i-get-pthreads-to-work-in-windows

const size_t M=1024*1024;

int cpu_loop_cnt=100;

void burn_cpu(int ntimes) {
   int j;     double x=2;
   printf("... cpu loop %d Mio times x=sin(x)\n",ntimes);
   for( j=0; j<ntimes; j++) {
       int i;
       for (i=0;i<M ; i++) x=sin(x);
   };
}

void my_thread(void *ptr) {
  burn_cpu(cpu_loop_cnt);
}
```

ganzes Programm (2/4)

```
void menu() {
   printf("\ncommands:\n");
   printf(" c 500      reserve 500 1M blocks with calloc\n");
   printf(" f          free last reserved buffer\n");
   printf(" w 1000     write 1000 times at random pos to buffer\n");
   printf(" l 10       loop cpu with 10 times 1 mio x=sin(x)\n");
   printf(" t 5        loop cpu with 5 threads, each %d times 1 mio x=sin(x)\n", cpu_loop_cnt);
   printf(" e          end program\n");
   printf("enter command:");
}

void threat_test(int n) {
   pthread_t thp[100] ;
   if (n>99) n=99;
   int i;
   for(i=0;i<n ; i++) {  // Threats anlegen und starten
      if ( pthread_create ( &thp[i] , NULL, (void *) &my_thread , NULL) ) perror ( "could not create thread" );
   }
   for(i=0;i<n ; i++) {
      pthread_join ( thp[i] , NULL) ;  // Warten bis threats fertig sind
   }
}
```

ganzes Programm (3/4)

```
int main() {
   size_t n, bsize=0,pos;
   while(1)  {
      menu();
      char command;        char zeile[100];  char *buffer, *l; int j;
      fgets(zeile,99,stdin);       sscanf(zeile,"%c %d", &command , &n);
     switch ( command)    {
       case 'c' :
          buffer=calloc(n, M);  bsize=n;
          printf("%5d 1M Bloecke reserviert ab Adresse: %p\n",n,buffer);
          printf("                        bis Adresse: %p\n",buffer+n*M);
         break;
        case 'w':
        case 'W':
         printf("fill %d times a random position with 1MByte of 0xff\n",n);
          for( j=0; j<n; j++) {
           pos= M*(rand()%bsize);  l=buffer+pos;
           if( command == 'W' ) printf("write at offset %10ld %p \n",pos,l);
           int i;            for (i=0;i<M ; i++) *(l++)=0xff;
          };
          break;
```

ganzes Programm (4/4)

```
    case 'f':
        free(buffer);
        break;
    case 'l':
        printf("cpu loop %d Mio times x=sin(x)\n",n);
        burn_cpu(n);
        cpu_loop_cnt=n;
        break;
    case 't':
        printf("cpu loop with %d threads %d Mio times x=sin(x)\n",n,cpu_loop_cnt);
        threat_test(n);
        break;
     case 'e':
          return(0);
      }
   }
   return 0;
}
```