- *int clone(int(*fn)(void*), void*child_stack, int flags, void *arg);*
- nützlich für Implementierung von Threads
- *clone()* erlaubt dem Kinderprozess, Teile seines Ausführungskontextes mit dem Elternprozess zu teilen(z.B. virtuelle Speicheradresse usw.)
- Erzeugen eines Kindes nach dem Aufruf von *fn(arg)*
- Der Kindprozess terminiert nach der Rückgabe der Funktion *fn(arg)*

- child thread PID
- Eltern- und Kinderprozess teilen…
- Erzugen des Kinderprozesses

- bei CLONE_PARENT | CLONE_FS | CLONE_FILES|... | CLONE_THREAD hat man das maximale ‘Sharing’

- Beispiel:
  #include

```
#include
#include
#include
#include
#include
#include

int variable, fd;

int do_something() {
   variable = 42;
   close(fd);
   _exit(0);
}

int main(int argc, char *argv[]) {
   void **child_stack;
   char tempch;

   variable = 9;
   fd = open("test.file", O_RDONLY);
   child_stack = (void **) malloc(16384); /*allocating space for the stack of
the new child prozess*/
   printf("The variable was %d\n", variable);

   clone(do_something, child_stack, CLONE_VM|CLONE_FILES, NULL); /*begin a new
context of execution with the given function*/
/* do_something = fn(arg)  pointer to a function*/
/* child_stack pointer to the stack space that was set up for child*/
/* third parameter – how much of the prozess context will be shared between the
child and the parent( share memory and share file descriptors)*/
/*NULL is our void *args → pointer to the arguments that will pass to the
function that the child process will execute */
   sleep(1);

   printf("The variable is now %d\n", variable);
   if (read(fd, &tempch, 1) < 1) {
```

```
      perror("File Read Error");
      exit(1);
   }
   printf("We could read from the file\n");
   return 0;
}
```

-