

int NumberOfSetBits(unsigned int i)

{

i = i - ((i >> 1) & 0x55555555);

i = (i & 0x33333333) + ((i >> 2) & 0x33333333);

unsigned split\_to\_4 = (((i + (i >> 4)) & 0x0F0F0F0F));

return ( ((split\_to\_4 >>24)&31) + ((split\_to\_4 >> 16)&31)

+ ((split\_to\_4 >> 8)&31) + ((split\_to\_4 )&31) );

}

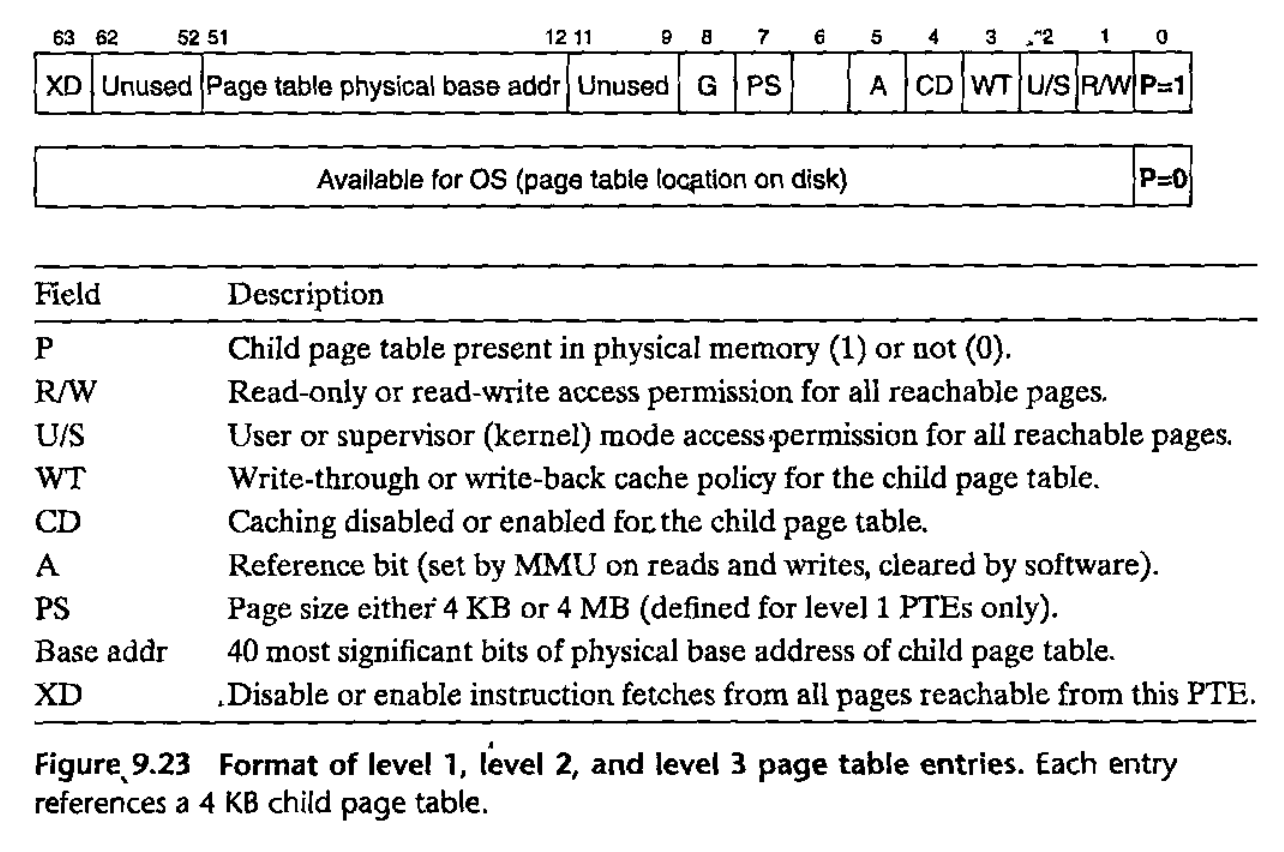
int main(int argc, char\* argv[]);

#include <[sys/mman.h](https://linux.die.net/include/sys/mman.h)>

void \*mmap(void \*addr, size\_t lengthint " prot ", int " flags ,

int fd, off\_t offset);

int munmap(void \*addr, size\_t length);



G = GLOBAL page (don’t evict from TLB on task switch)

Level 4 has DIRTY BIT at 6 and PS=0 (PS only defined for L1)

L1,2,3: When P=1 (always for Linux), address field is 40-bit PPN -> beginning of page table in level +1

Level 4: 40-bit PPN points to base of PHYSICAL PAGE. 4KB alignment for physical page and page tables.

CR3 register contains PHYSICAL address of THE L1 page table. CR Control Register

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#define FILEPATH "/tmp/mmapped.bin"

#define NUMINTS (1000)

#define FILESIZE (NUMINTS \* sizeof(int))

int main(int argc, char \*argv[])

{

int i;

int fd;

int result;

int \*map; /\* mmapped array of int's \*/

/\* Open a file for writing.

\* - Creating the file if it doesn't exist.

\* - Truncating it to 0 size if it already exists. (not really needed)

\*

\* Note: "O\_WRONLY" mode is not sufficient when mmaping.

\*/

fd = open(FILEPATH, O\_RDWR | O\_CREAT | O\_TRUNC, (mode\_t)0600);

if (fd == -1) {

perror("Error opening file for writing");

exit(EXIT\_FAILURE);

}

/\* Stretch the file size to the size of the (mmapped) array of ints

\*/

result = lseek(fd, FILESIZE-1, SEEK\_SET);

if (result == -1) {

close(fd);

perror("Error calling lseek() to 'stretch' the file");

exit(EXIT\_FAILURE);

}

/\* Something needs to be written at the end of the file to

\* have the file actually have the new size.

\* Just writing an empty string at the current file position will do.

\*

\* Note:

\* - The current position in the file is at the end of the stretched

\* file due to the call to lseek().

\* - An empty string is actually a single '\0' character, so a zero-byte

\* will be written at the last byte of the file.

\*/

result = write(fd, "", 1);

if (result != 1) {

close(fd);

perror("Error writing last byte of the file");

exit(EXIT\_FAILURE);

}

/\* Now the file is ready to be mmapped.

\*/

map = mmap(0, FILESIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (map == MAP\_FAILED) {

close(fd);

perror("Error mmapping the file");

exit(EXIT\_FAILURE);

}

/\* Now write int's to the file as if it were memory (an array of ints).

\*/

for (i = 1; i <=NUMINTS; ++i) {

map[i] = 2 \* i;

}

/\* Don't forget to free the mmapped memory

\*/

if (munmap(map, FILESIZE) == -1) {

perror("Error un-mmapping the file");

/\* Decide here whether to close(fd) and exit() or not. Depends... \*/

}

/\* Un-mmaping doesn't close the file, so we still need to do that.

\*/

close(fd);

return 0;

#include *<stdio.h>*

#include *<setjmp.h>*

**static** jmp\_buf buf;

void second(void) {

printf("second**\n**"); *// prints*

longjmp(buf,1); *// jumps back to where setjmp was called - making setjmp now return 1*

}

void first(void) {

second();

printf("first**\n**"); *// does not print*

}

int main() {

**if** (!setjmp(buf))

first(); *// when executed, setjmp returned 0*

**else** *// when longjmp jumps back, setjmp returns 1*

printf("main**\n**"); *// prints*

**return** 0;

}

**ERRNO** is a thread local storage (global variable unique to each thread) variable that can be used to give a description of what went wrong.

Sometimes we want it such that when something goes wrong we return back to a specific function. (like try-catch-throw blocks)

int **setjmp**(jmp\_buf env) and void **longjmp**(jmp\_buf env, int retval) take care of this.

**Software Signal Handler (software version of exceptions)**

void(\***signal**(intsig,void(\*func)(int)))(int);

**Exceptions** are Hardware based and move control to Kernel

**-Interrupts**: Most commonly from I/O device; **ASYNCHRONOUS**.

§ Trigger interrupt pin -> stop execution -> switch to INTERRUPT HANDLER.

§ This happens within the same thread, and control is handed to kernel.

**-Trap**: Do things not within the scope of program and need kernels help; **SYNCHRONOUS**

-**Fault**: Caused by potentially recoverable but unexpected errors (like Page Fault); **SYNCHRONOUS**

**-Abort**: Unrecoverable fatal error such as corrupted memory or hardware error. Exit.

**Sync** if triggered by MMU (eg. tried to write in read-only); **Async** if RAM breaks or smth.