Una rezon por la que un programa crashes es que no hay memoria suficiente, ósea esta accediendo a memoria inválida o sea, una porción de la memoria que no está asignada a ese proceso. Normalmente esto pasa con otros lenguajes de programación cuando el programador necesita especificar cual es la porción de la memoria que el proceso va a utilizar.

Parar tener más información podemos usar un debugger para saber que es lo que esta pasando en el proceso, y con esto podemos arreglarlo.

Ahora, mira esto:

Si no es suficiente con esto, vamos a necesitar que el programa este compilado, a parte de la compilacion normal binaria, con Debugging symbols, estos contienen las funciones que están siendo usadas, los nombres de estas, los nombres de las funciones y los resultados o outputs.

Así que, vamos o numero uno tener que recompilar el código para poder incluir los Debugging symbols con más detalles, o descargas los error symbols al proveedor del software

Con los Debugging symbols, debemos usar un debugger.

**Undefined behavior:**

Esto significa que el código esta haciendo algo que no es valido en el lenguaje de programación.

**Valgrind: Linus, MacOs**

**Dr. Memory: Windows, Linux**

Es una herramienta que nos permite saber si el código esta haciendo alguna actividad invalida, sin importar si este crashea o no. Este accede a las variables antes del código y se asegura de que estas sean liberadas de la memoria una vez que ya no son utilizadas, o si el código esta apuntando a una parte de la memoria que es inválida.

Unhandled Errors and Exceptions:

Cuando un programa muestra un error normalmente nos de el tipo de error, la línea en la que ocurrió el error y el Traceback, el Traceback muestra las líneas de las diferentes funciones que estaban siendo ejecutadas al momento que el problema sucedió

Hay que tener cuidado porque muchas veces el error que se muestra no tiene nada que ver con lo que realmente está pasando porque el error se puede dar en una función que esta usando una variable que no esta bien inicializada, y la variable no esta dando error, sino la función.

Para esto, podemos usa un debugger, con lo que podemos hacer nuestro trabajo mucho mas simple cuando tenemos mucho código y el problema no está a la vista.

Para los archivos .py, ósea, para el lenguaje de programación python podemos usar la herramienta BDB que es in debugger interactivo que nos permite hacer todas las acciones típicas de un debugger.

Código de otra persona:

Es básico, si hay un problema en una función, no debemos enfocarnos en todo el código, hay que buscar la raíz, la raíz de la raíz, e incluso la raíz de la raíz de la raíz de la raíz. Esto significa, que si hay una función que esta crasheando, debemos ver que otras funciones llama, si el resultado devuelto por esa función es el esperado, o si el eso pasa con una función que llama a otra función que llama a otra función, parece demasiado, pero revisar 10 funciones en un código de 10k líneas, es una mejor muy grande. Si esa funcione esta crasheando, el bug jamás va a estar en un dato que es llamado por esa función.

Por es muy importante la manera de la que yo escribo el código, tratando de que no crashee, con data o input no deseado, moldeándolo para que, de un output, o devolviendo prints que expliquen exactamente que es lo que pasa, para que el usuario no vea un error, y el programa crashee, sino que devuelve un output, y el usuario es capaz de continuar usando el programa.

Core Files:

Estos ficheros almacenan toda la información relacionada don el crash para que alguien mas lo pueda corregir los bugs en el código. Para crear un core file, lo único que tenemos que hacer, al menos en Linux, es pasarle el comando **unlimit -c unlimited**, y en la siguiente línea del promt, ejecutamos el script que esta dando el error, en este caso, un segmentation fault

Una vez que tenemos el core file, podemos usarlo pasándoselo a un debugger, por ejemple el **gdb** pasándole los siguientes argumentos **gdb -c core example**, en donde example, es el nombre del fichero que crasheó.

Esto va a abrir un enviroment, como el de python que nos devuelve >>>, este enviroment lo que dos devuelve es (gdb) y en este podemos usar comandos que sean específicos de gdb, por ejemplo:

**backtrace:** Este es el que nos devuelve la raíz de la raíz de la raíz de la raíz, con este podemos ver quien está crasheando y de donde viene, permitiéndonos rastrear la verdadera raíz.

**up** : Este comando lo podemos usar después de haber utilizado el comando backtrace, este nos permite seguir el árbol de dicho error, y ver que función está llamando a que función y así sucesivamente.

**list**: Este nos enseña en el código, cuáles son las líneas que esta involucradas, no solo nos dice cuales fueron, sino que nos muestra el código como tal para que podamos ver