

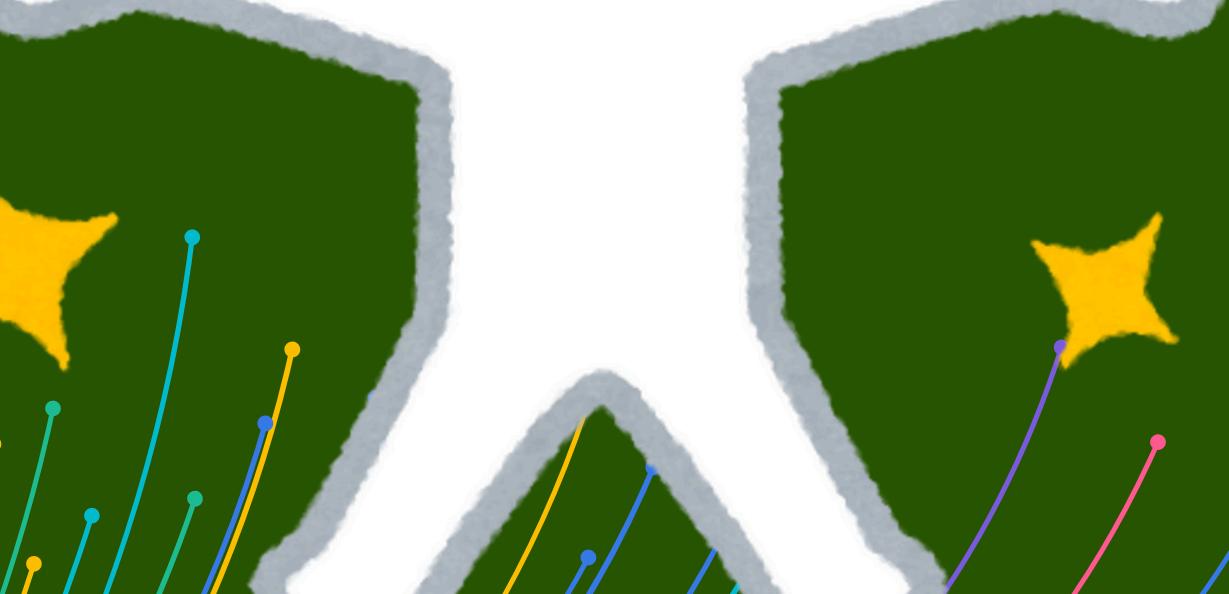
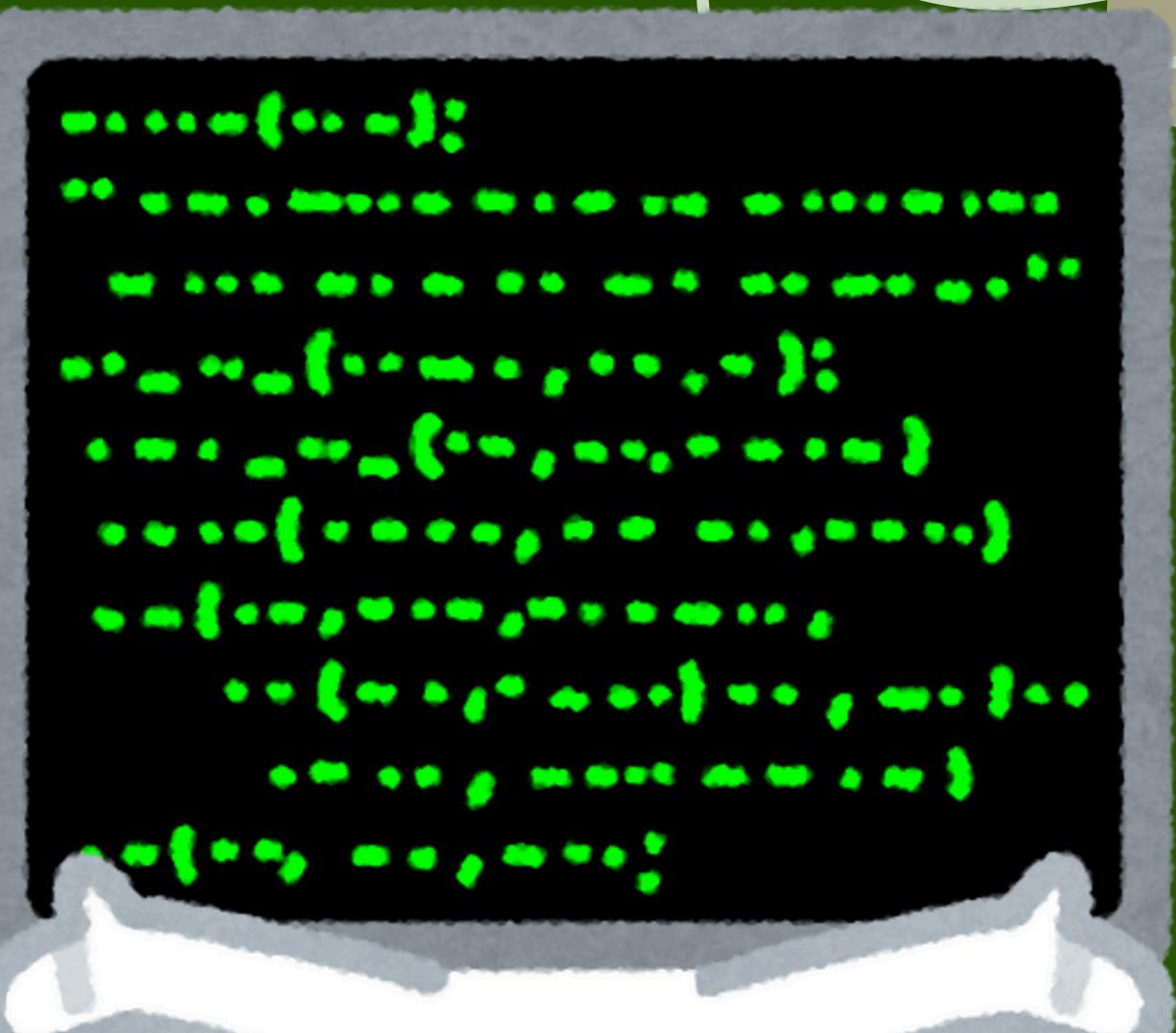
**Sistemas Operativos**

# Ayudantía 1: Syscalls y fork

Profesor: Victor Reyes

Ayudante: Diego Banda

Sección 2



# Contacto



darklouds



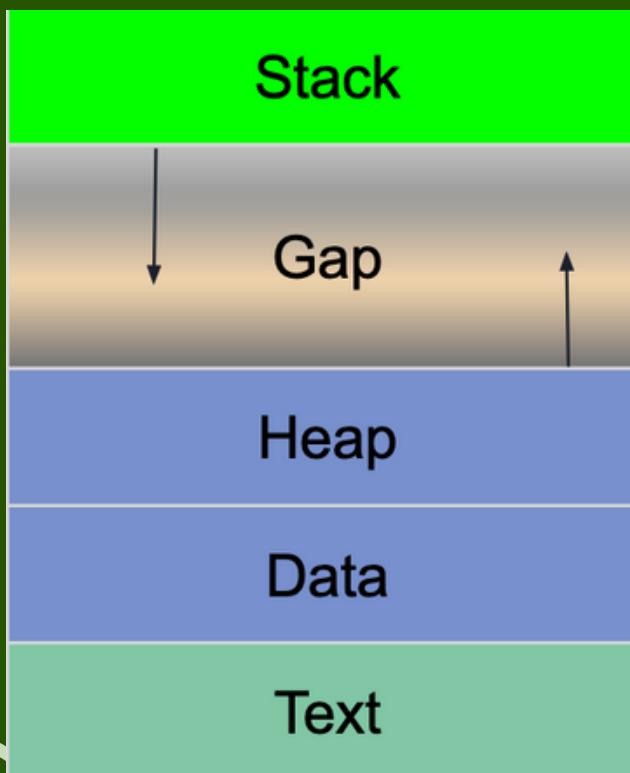
diego.banda@mail\_udp.cl



# Procesos

Es todo, literalmente todo, lo que está ejecutándose en el computador, desde el Sistema Operativo hasta un programa o juego. Hay muchos procesos ejecutándose a la vez, y no existen tantos núcleos como para suplir cada uno, ¿Como lo hace entonces? Se “intercalan”

Para “intercalarse”, el computador debe tener guardada información clave de cada proceso:

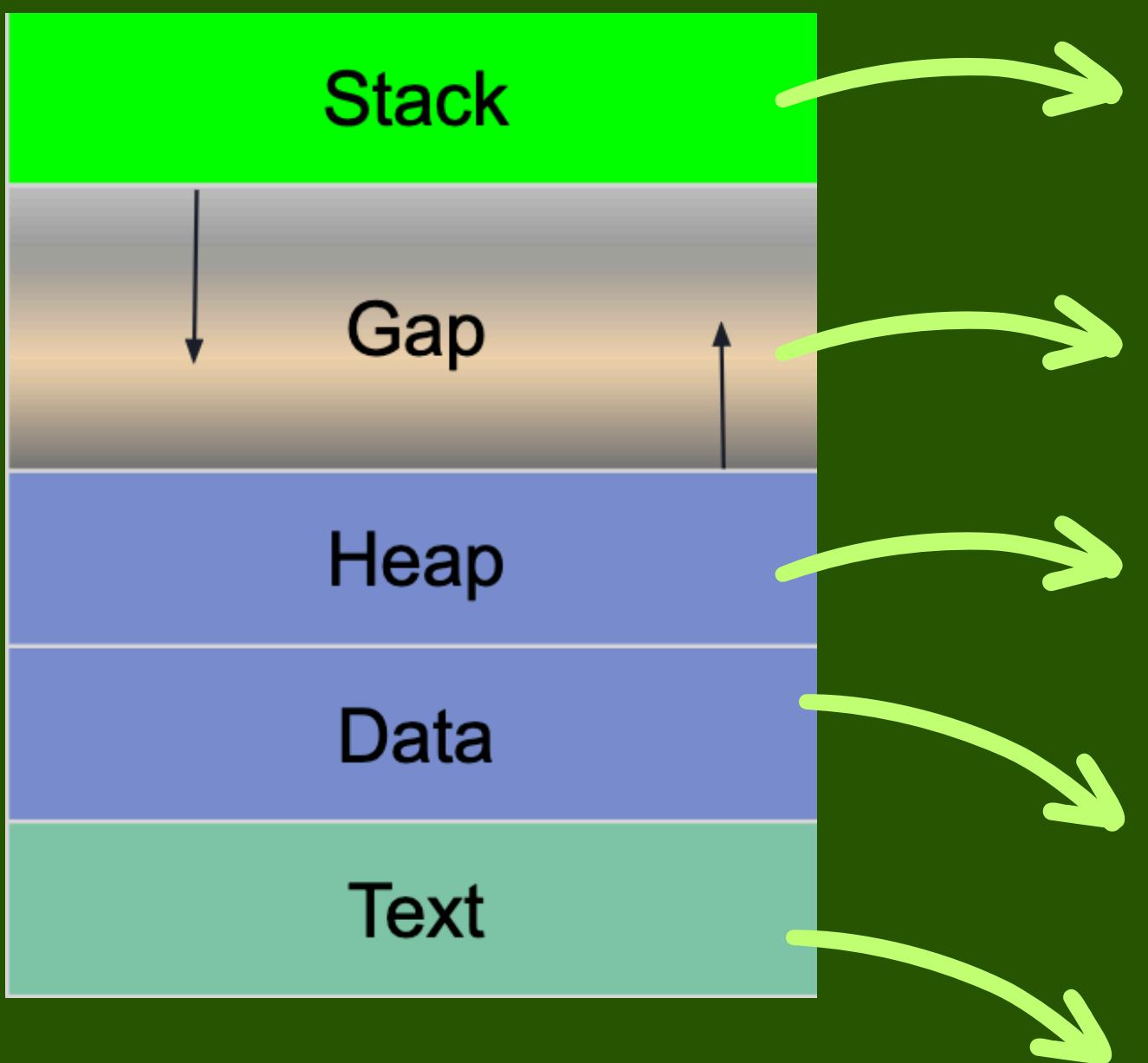


**Proceso en memoria**

Estado del proceso
Número del proceso
Program counter
Límites de memoria
Lista de archivos abiertos
...

**Process Control Block (PCB)**

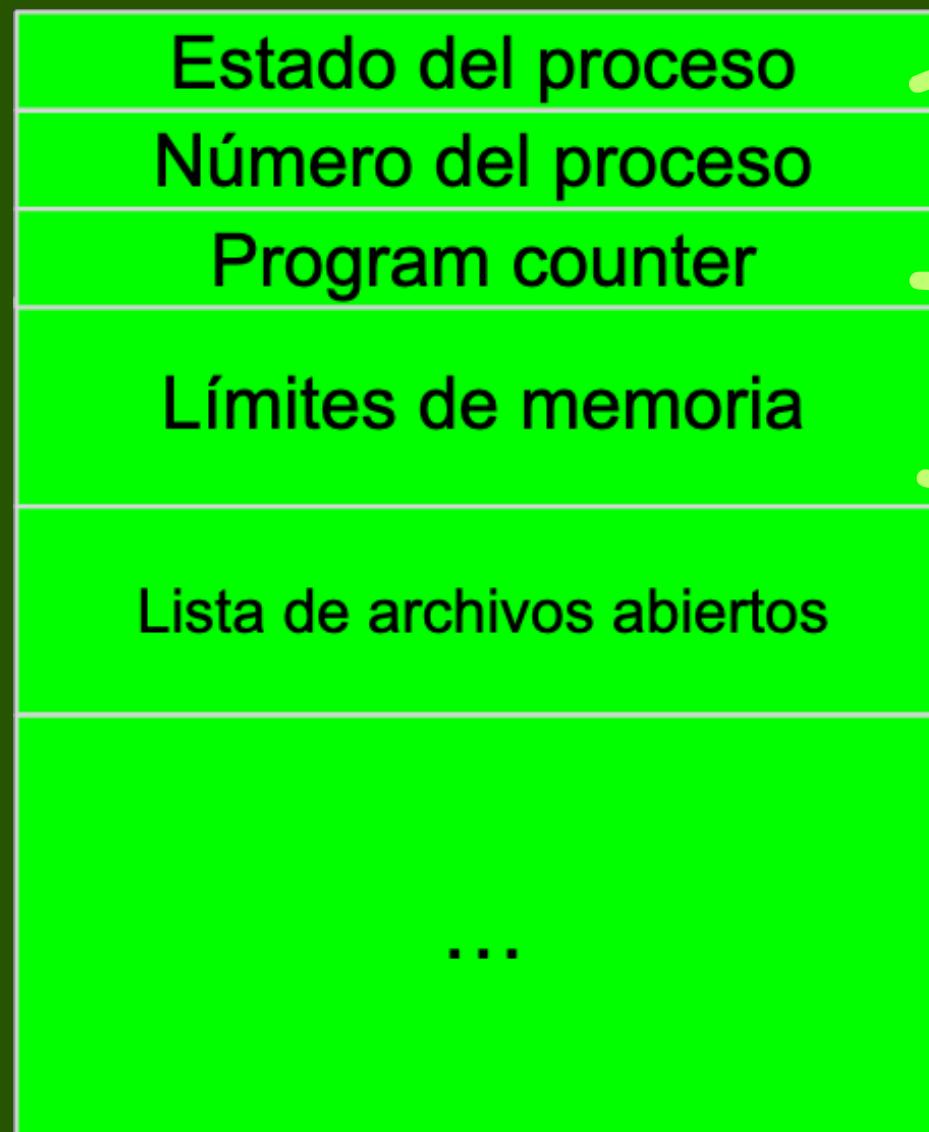
# Procesos



- Guarda las variables locales entre otros registros, crece hacia abajo.
- Espacio entre Stack y Heap hecho para crecer según lo necesario.
- Se usa como memoria dinámica en tiempo de ejecución (para new en C++ por ejemplo), crece hacia arriba.
- Variables globales y/o estáticas ya inicializadas.
- El código ejecutándose en el proceso.

# Procesos

## Process Control Block (PCB)



Estado actual (running, ready, waiting, terminated).

Identificación unica Process ID (PID).

Dirección de la proxima instrucción a ejecutar.

Límites de memoria asignada al proceso.

# Syscalls

Enlace entre modo usuario y modo kernel, permite al usuario solicitar un servicio que entrega el Sistema Operativo

OJO: Varían según SO.

Algunas de las más importantes:

- `fork()`: Bifurcación, crea una copia del proceso padre, con las mismas funciones, variables, espacio en memoria, etc.
- `wait()`: Un proceso padre espera la finalización de uno de sus procesos hijos.
- `exit()`: Termina la ejecución del programa.
- `sleep()`: Suspende la ejecución de un programa durante un tiempo determinado.
- `getpid()`: Retorna el process ID del actual proceso.

# Fork

Genera una bifurcación, creando un proceso hijo que es una copia del actual proceso, retornando un pid que puede significar varias cosas.

- $\text{pid} > 0$ : Estamos en el proceso padre y el pid es el process id del proceso hijo.
- $\text{pid} = 0$ : Estamos en el proceso hijo.
- $\text{pid} < 0$ : Estamos en el proceso padre y hubo un error al crear el proceso hijo.



De hecho

# Ejercicios

1.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(){
6     pid_t pid;
7
8     for(int i = 0 ; i < 5 ; i++){
9         pid = fork();
10        if(pid == 0){
11            break;
12        }else if(pid < 0){
13            break;
14        }
15    }
16 }
```



# Ejercicios

2.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(){
6     pid_t pid;
7
8     for(int i = 0 ; i < 5 ; i++){
9         pid = fork();
10        if(pid > 0){
11            break;
12        }else if(pid < 0){
13            break;
14        }
15    }
16 }
```



# Ejercicios

3.

Suponga que ejecutamos el siguiente código en C:

```
1 int num = 1; // variable global
2 int main(){
3     pid_t t = fork();
4     if (t != 0){
5         num = num*2;
6         pid_t tt = fork();
7         if (tt > 0){
8             num = num-2;
9         }
10        else if (tt < 0){
11            fork();
12            num = num+2;
13        }
14    }
15    num = num+1;
16    sleep(1);
17    printf("%d\n",num);
18 }
```

- ¿Cuántos procesos se crean en total? Explique.
- Indique 2 posibles salidas. Explique.
- Suponga ahora que por una razón misteriosa la primitiva fork( ) de la línea 6 falla para todos los posibles procesos que ejecutan dicha instrucción. ¿Cuál(es) sería(n) la(s) posible(s) salida(s)? Explique.

# Ejercicios

4.

Dado el siguiente código en C:

```
1 int main() {
2     pid_t t = fork();
3     int i = 2;
4     int status = -1;
5     if (t > 0){
6         status = 2;
7         i = 6;
8         wait(&status);
9     }
10    else{
11        if (i > 5){
12            exit(2);
13        }
14        else{
15            exit(3);
16        }
17    } // continua al lado
```

```
1     if (WEXITSTATUS(status) == -1){
2         printf("Los\n");
3     }
4     if (WEXITSTATUS(status) == 2){
5         printf("Pollos\n");
6     }
7     if (WEXITSTATUS(status) == 3){
8         printf("Hermanos\n");
9     }
10 }
11 }
```

Determine la(s) posible(s) salida(s). Explique.