



# Trabajo Obligatorio Sistemas Operativos III

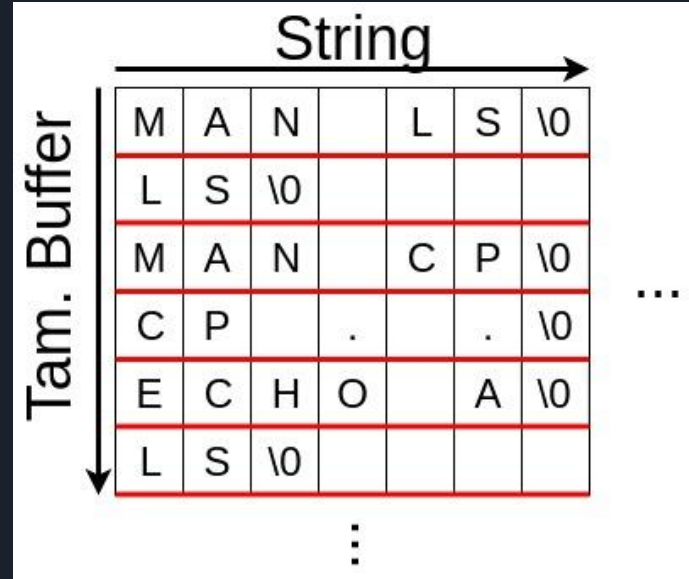
Emiliano Fernandez  
Juan Aparicio



# Solución al problema

# Buffer - Estructura

```
typedef struct {  
    char buffer[TAM_BUFFER][TAM_STRING];  
    int inicio;  
    int fin;  
    int largo;  
} buffer_t;
```





# Buffer - Métodos

```
void buffer_init(buffer_t *b);  
  
int buffer_push(buffer_t *b, string_p_t dato);  
  
int buffer_pop(buffer_t *b, string_p_t dato);
```



# Memoria compartida - Concurrency

```
typedef struct{
    buffer_t bufferCirc;
    int finArchivo;
}datos_sh_mem_t;
```


```
...
/*Inicio seccion critica*/
sem_wait(semaforo);
int finalizado = datos_sh_mem_finalizado(shMemData);
sem_post(semaforo);
/*fin seccion critica*/
...
...
/*Inicio seccion critica*/
sem_wait(semaforo);
buffer_push(&(shMemData -> bufferCirc), linea);
sem_post(semaforo);
/*Fin seccion critica*/
...
```



# Ejecución de los procesos

```
FILE *popen(const char *command, const char *type);
```

[git.musl-libc.org/cgit/musl/tree/src/stdio/popen.c](https://git.musl-libc.org/cgit/musl/tree/src/stdio/popen.c)



# Asignación de memoria y almacenamiento secundario

Demo



# Solución alternativa con mensajería

```
struct mymsg {  
    long    mtype;  
    char    mtext[1];  
}
```

```
int msgget(key_t key, int msgflg);
```

```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

```
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
```





# Conclusiones

- Se desperdicia mucha memoria con la memoria compartida.
- Se hacen muchos accesos a memoria por el polling.
- El proceso consumidor queda en ejecución todo el tiempo (`while (!finalizado){...}`), consumiendo así mucho CPU.
- Una mejor solución se podría hacer con mensajería.