Politecnico di Torino
Programmazione di Sistema

A.A. 2021/2022

Report progetto C2
(SHELL)

Gruppo
Bruno Giacomo - 301311
Ciacco Giuseppe - 295982
Ciravegna Flavio - 303398

Report progetto C2

1 Introduzione

Per la realizzazione del progetto è stata utilizzata come base la versione di os161 contenente le soluzioni dei laboratori visti a lezione. Come richiesto è stata creata l'opzione SHELL per abilitare le parti di codice aggiunto rispetto la versione originale. In particolare sono stati aggiunti i file opzionali dir_syscalls.c, file_syscalls.c e proc_syscalls.c.

Tra le modifiche effettuate al codice del laboratorio è presente l'aggiunta del controllo degli errori, seguendo le indicazioni dei manuali di os161 e Linux. A quelli già presenti in errno.h sono stati aggiunti EOVERFLOW ed ETXTBSY.

Un'altra modifica da segnalare è l'incremento della ram (disponibile in sys161.conf), la quale è stata portata da 512K a 16M.

Alle syscall richieste sono state aggiunte la fstat e la getdirentry per il funzionamento del comando ls eseguendo bin/sh.

Eseguendo il comando "bin/sh" sono disponibili i comandi: **cat**, **cp**, **false**, **true**, **pwd**, **ls** (quest'ultimo non richiesto nella traccia del progetto, implementato per facilitare i test sulla shell)

2 Implementazione system calls

Le syscall implementate condividono la medesima struttura, ovvero:

- Verifica di validità dei parametri
- Allocazione e inizializzazione di eventuali strutture dati necessarie per l'esecuzione della syscall
- Esecuzione delle operazioni specifiche della syscall (nel caso delle syscall su file e direttori, chiamata a funzioni del tipo VOP_ o vfs_)
- Gestione del valore di ritorno e eventuale deallocazione delle strutture dati

2.1 open

Per la realizzazione della open è stata utilizzata la soluzione del lab5, la quale poi è stata modificata per ottenere un'implementazione più completa. Segue la struttura standard delle syscall implementate, con chiamata alla vfs_open per eseguire le operazioni specifiche della syscall.

Errori implementati

- **OS161:** EFAULT, EINVAL, ENOTDIR, EISDIR, ENOENT, EEXIST, ENODEV, ENXIO, EMFILE, ENFILE, EIO, ENOSPC
- **Linux:**
  - ENOMEM: errore di allocazione delle strutture dati interne
  - ENAMETOOLONG: lunghezza path maggiore lunghezza massima

2.2 read/write

L'implementazione della read è stata realizzata a partire dalla soluzione proposta per i laboratori relativi a tale syscall. Nella versione di partenza input e output su STDIN, STDOUT e STDERR veniva gestita in modo differente rispetto a file veri e propri. Nell'implementazione corrente non è presente una distinzione esplicita e tutte le operazioni di I/O sono gestite allo stesso modo. Si è infatti constatato che la VOP_READ e la VOP_WRITE permettono di effettuare operazioni di I/O sui file descriptor corrispondenti alla console (con:) senza necessità di chiamate esplicite a getch e putch. Le funzioni sys_read e sys_write sono wrapper per le rispettive funzioni file_read e file_write. Inoltre nell'implementazione originale non veniva gestito l'accesso simultaneo allo stesso file (in particolare

in caso di lettura dalla console; questo risultava particolarmente problematico nell'utilizzo della shell in cui sia il menu di os161 che la shell rimanevano in attesa di input, rendendo necessario il dover scrivere ogni carattere due volte per eseguire correttamente un comando sulla shell). Il problema è stato risolto mediante sincronizzazione tramite lock per garantire letture corrette da parte dei processi.

- **file_read**
  Segue la struttura standard delle syscall implementate, con chiamata alla VOP_READ per eseguire le operazioni specifiche della syscall.

  Errori implementati

  - **OS161:** EFAULT, EBADF, EIO
  - **Linux:**
    * EISDIR: tentativo di lettura su directory

- **file_write**
  Segue la struttura standard delle syscall implementate, con chiamata alla VOP_WRITE per eseguire le operazioni specifiche della syscall.

  Errori implementati

  - **OS161:** EFAULT, EBADF, EIO, ENOSPC
  - **Linux:**
    * EFBIG: Errore di scrittura dovuto a dimensioni del file da scrivere

2.3 lseek

Segue la struttura standard delle syscall implementate, con chiamata alla VOP_ISSEEKABLE per eseguire le operazioni specifiche della syscall.

La VOP_ISSEEKABLE utilizza funzioni di emufs non implementate del tutto (emufs_isseekable()), a causa delle quali non viene ritornato l'errore ESPIPE. Questo avverrebbe nel caso si chiamasse la lseek su pipe, socket o FIFO. Non essendo queste argomento del progetto, si è deciso di non implementarla. Nonostante ciò, per completezza della funzione sys_lseek, si è scelto di mantenere il check sul vnode effettuato mediante VOP_ISSEEKABLE.

Errori implementati

- **OS161:** EINVAL, EBADF, ESPIPE (gestita ad alto livello, non in VOP_ISSEEKABLE)
- **Linux:**
  - EOVERFLOW: l'offset del file non è rappresentabile in una variabile off_t

2.4 close

La syscall close è stata ultimata modificando la versione "base" realizzata durante i laboratori, il cui scopo di base è quello di chiudere il file handle (fd) passato come parametro.

L'accesso concorrente alle strutture dati condivise (come la file table del processo corrente o la struttura openfile) è stato protetto da appositi lock.

Errori implementati

- **OS161:** EBADF, EIO

2.5 dup2

Segue la struttura standard delle syscall implementate, con accesso alla fileTable ai file descriptor corrispondenti a oldfd e newfd per eseguire le operazioni specifiche della syscall. A differenza di Linux, vengono riportati anche gli eventuali errori generati in fase di chiusura di newfd.

Errori implementati

- **OS161:** EBADF, EIO

Errori non implementati (OS161)

- EMFILE: non sono presenti vincoli sui file apribili per processo
- ENFILE: non sono presenti vincoli sui file apribili a livello di sistema

2.6 chdir

Segue la struttura standard delle syscall implementate, con chiamata alla vfs_open e alla vfs_setcurdir per eseguire le operazioni specifiche della syscall.

Basandosi interamente su vfs_open e vfs_setcurdir (nel caso di emufs, tali funzioni chiamano rispettivamente emufs_open e emufs_setcurdir, entrambe già implementate), l'implementazione non ha richiesto ulteriori modifiche su altri files.

Errori implementati

- **OS161:** EFAULT, ENOTDIR, ENOENT, ENODEV, EIO
- **Linux:**
  - ENOMEM: errore di allocazione delle strutture dati interne
  - ENAMETOOLONG: lunghezza del path maggiore della lunghezza massima

2.7 __getcwd

Segue la struttura standard delle syscall implementate, con chiamata alla vfs_getcwd per eseguire le operazioni specifiche della syscall.

A differenza delle altre syscalls di I/O, per la getcwd è stato necessario effettuare modifiche anche in emu.c implementando la emufs_namefile.

Nella versione iniziale infatti la emufs_namefile permette di ottenere un risultato valido solo se la current directory è la root del filesystem (nel caso di emufs emu0:), ritornando errore in tutti gli altri casi. Si è pertanto implementata la emufs_namefile per gestire la navigazione nei direttori e ottenere il path completo a partire da un vnode. In questo modo si è voluta fornire una più completa implementazione della getcwd.

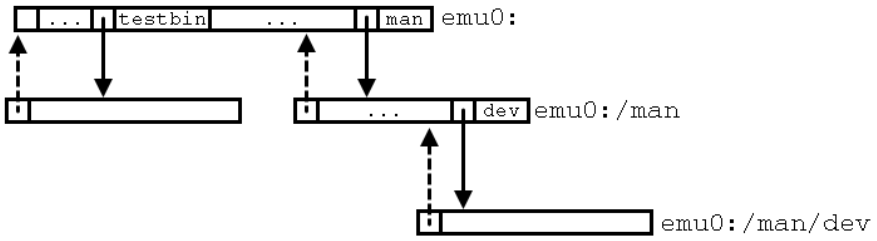
Dal momento che il nome relativo a un vnode è contenuto nel vnode padre e non nel vnode stesso (in modo analogo agli inode linux) è necessario effettuare una scansione sul vnode padre fino alla radice. Per ovviare a questa problema si sarebbe potuto tenere traccia della directory corrente aggiornando una variabile, utilizzata come stato, a ogni chiamata della chdir.

Si è preferito utilizzare una soluzione in cui il path viene generato come stringa a ogni chiamata della getcwd che, pur necessitando di maggiori operazioni rispetto alla versione con uno stato, ricalca un funzionamento più simile a quello della getcwd Linux.

Errori implementati

- **OS161:** EFAULT, EIO

- **Linux:**
  - ENOMEM: errore di allocazione delle strutture dati interne
  - ENAMETOOLONG: lunghezza del path maggiore della lunghezza massima
  - EINVAL: dimensione del buffer (passata come parametro) nulla
  - ERANGE: implementato in emufs_namefile, buffer di dimensione non sufficiente per il path



Gerarchia vnode

2.8 getpid

Essendo una syscall semplice (ritorna unicamente il process id del processo corrente), non presenta modifiche rispetto alla versione dei laboratori. Inoltre - così come riportato dal Reference Manual - non include errori di ritorno.

2.9 fork

Per l'implementazione della fork è stata completata la versione preliminare fornita con le soluzioni del laboratorio 4 di OS161. Sostanzialmente, al fine di permettere un corretto funzionamento della syscall "waitpid", è stato necessario effettuare il link tra processo padre e figlio nella struct proc (come citato nella descrizione della waitpid).

Il corretto funzionamento della fork (e della relativa gestione degli errori) è stato testato utilizzando testbin/forktest (esito in appendice) e testbin/bigfork, il quale viene completato con successo in tutti e 5 gli stages.

Errori implementati

- **OS161:** ENOMEM, ENPROC

Errori non implementati (OS161)

- EMPROC: gestione utenti non implementata in OS161

2.10 execv

Per realizzare la execv si è utilizzato come base di partenza la funzione runprogram in runprogram.c. A questa sono stati aggiunti i check sulla validità dei parametri ricevuti, poi si è copiato il nome del programma in diverse stringhe per aggiornare i nomi del thread e del processo attualmente in esecuzione. In seguito si apre il file corrispondente (come in runprogram). Fatto questo si procede alla copia degli argomenti dallo stack user al vettore kargbuf mediante la funzione copy_args. Facendo questo lo stack user non è più necessario e si procede alla sostituzione dell' addrspace del processo

con uno nuovo, nel quale verrà caricato l'eseguibile tramite la load_elf e definito un nuovo stack. Una volta disponibile l'indirizzo del nuovo stack si procede al'aggiormento dei puntatori alle stringhe contenenti gli argomenti e poi alla copia di kargbuf nello stack user. Al termine si deallocano le strutture non più necessarie e si chiama la enter_new_process.

- **copy_args**: funzione che riceve tre parametri: uargs (puntatore al vettore degli argomenti in memoria user), nargs (puntatore a variabile intera nel quale verrà ritornato il numero di argomenti passati [argc]), buflen (puntatore alla variabile intera nella quale verrà ritornata la lunghezza in byte del vettore dei puntatori + stringhe degli argomenti) In una prima parte si esegue un ciclo while nel quale viene copiata una stringa alla volta in memoria kernel per poi aggiornare il numero di argomenti e la loro lunghezza. Per la lunghezza delle stringhe si considera l'approssimazione per eccesso al più vicino multiplo di 4, questo causa dell'allineamento su 32 bit richesto dall'architettura MIPS. Nella seconda parte si procede alla creazione del vettore di puntatori in kargbuf (in questo momento ridotti solo all'indice di partenza della stringa in kargbuf) e alla copia dei vari argomenti.
- **adjust_kargbuf**: funzione che riceve due parametri: n_params (argc) e stack_ptr (indirizzo a partire dal quale verrà copiato kargbuf). Si aggiorna il vettore di puntatori sommando ai vari offset l'indirizzo di partenza nello stack per ottenere i valori degli indirizzi che le stringhe avranno una volta copiate nella memoria user. (diagrammi in appendice)

Errori implementati

- **OS161:** ENOMEM, EFAULT, ENOEXEC, E2BIG, ENOTDIR, EISDIR, ENOENT, ENODEV, EIO
- **Linux:**
  - ENAMETOOLONG: lunghezza del path maggiore della lunghezza massima

2.11 waitpid

La versione di partenza della waitpid è quella fornita come soluzione del laboratorio 4. Al fine di una completa implementazione di questa syscall, sono stati aggiunti i seguenti campi alla struct proc (in kern/include/proc.h):

- int p_exited, che indica se il processo è gia uscito
- struct proc parent_proc, il quale fa riferimento al processo parent

Oltre a questo, l'accesso alle strutture dati condivise è stato gestito mediante l'acquisizione e rilascio degli opportuni spinlock.

Il generale funzionamento di questa syscall prevede di, inizialmente, invocare la proc_search_pid al fine di ottenere il puntatore alla struct proc identificata dal pid passato come parametro. Vengono poi effettuati i vari controlli sulla presenza di eventuali errori.

Il cuore della waitpid() consiste nella chiamata alla "proc_wait", a meno che l'option passata come parametro non sia "WNOHANG" la quale - come riportato dal Reference Manual - "causes waitpid, when called for a process that has not yet exited, to return 0 immediately instead of waiting". Infine, in caso di successo, viene ritornato il process id.

Errori implementati

- **OS161:** EFAULT, EINVAL, ESRCH, ECHILD

2.12 _exit

La syscall exit non presenta modifiche strutturali rispetto alla versione implementata nei laboratori. Si è modificata la gestione del valore di ritorno secondo lo standard di os161, mostrato nel paragrafo 3, relativamente alla kill_curthread.

Si è implementata la proc_rm_parent_link per gestire la modifica del puntatore al processo padre nel caso in cui quest'ultimo termini prima dei processi figli. Non essendo presente una lista dei processi figli nella struct proc, si è optato per una soluzione più semplice in cui viene scandita la tabella dei processi cercando quelli con puntatore al processo terminato.

2.13 fstat

Non indicata tra le syscall da implementare. Implementata per il corretto funzionamento della ls in bin/sh (utilizzata principalmente per facilitare i test sulla shell) Segue la struttura standard delle syscall implementate, con chiamata alla VOP_STAT per eseguire le operazioni specifiche della syscall.

Errori implementati:

- **OS161:** EFAULT, EBADF, EIO

2.14 getdirentry

Non indicata tra le syscall da implementare. Implementata per il corretto funzionamento della ls in bin/sh (utilizzata principalmente per facilitare i test sulla shell) Segue la struttura standard delle syscall implementate, con chiamata alla VOP_GETDIRENTRY per eseguire le operazioni specifiche della syscall.

Errori implementati:

- **OS161:** EFAULT, ENOTDIR, EBADF, EIO

3 Ulteriori modifiche al kernel

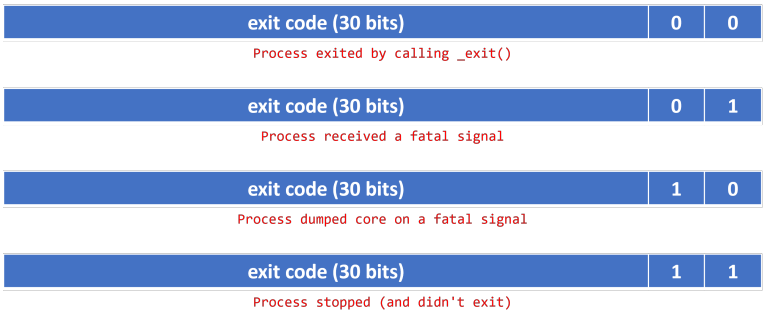
- **syscall.c:** sono stati aggiunti nello switch i case corrispondenti alle syscalls implementate. Esse contengono tra i parametri ricevuti un puntatore a intero nel quale verrà ritornato il codice dell'errore (se necessario). Le uniche eccezioni sono la sys_exit e la sys_getpid le quali non ritornano errori, e la sys_fork la quale ha come valore di ritorno il codice di errore mentre il pid è passato attraverso puntatore ricevuto come parametro. Un caso particolare è sys_lseek in quanto off_t è di 64 bit anzichè 32. Per questo motivo si sono realizzate 3 macro per la gestione dei registri e la conversione da 32 a 64 e viceversa. Queste sono la MAKE_64BITS la quale riceve due valori su 32 bit e ne forma uno su 64, la GET_LO che ritorna gli ultimi 32 bit del valore e la GET_HI che ne ritorna i 32 bit alti.
- **dumbvm.c:** In dumbvm.c è stata aggiunta la funzione is_valid_pointer che verifica se l'indirizzo addr è contenuto nel address space passato come parametro.
- **proc.c:** Alla processTable è stato aggiunto il campo booleano is_full per indicare se la tabella dei processi è piena, evitando così di chiamare un panic che bloccherebbe l'intero sistema. Per leggerne il valore si è implementata la funzione is_proc_table_full.
- **synchronization:**
  - Alla struct proc è stato aggiunto il lock ft_lock per proteggere la fileTable del processo. Questo verrà creato e distrutto insieme al processo.

  - Alla struct openfile è stato aggiunto il campo of_lock per proteggere la singola struttura senza bisogno di possedere il lock dell'intera tabella. Si ha la creazione e la distruzione durante rispettivamente la open e la close del file.
  - In proc.c è stato aggiunto il lock ft_copy_lock, necessario nella funzione proc_file_table_copy per evitare deadlock causati da un accesso a due lock su due file table differenti. Per evitarli dunque si acquisisce prima il lock per la copia e in seguito avviene l'acquisizione dei lock sulle tabelle. Tale lock viene inizializzato al bootstrap nella funzione proc_bootstrap
  - In file_syscalls.c la system file table è stata implementata con una struct contenente il vettore di struct openfile e un lock per la tabella. Quest'ultimo verrà inizializzato mediante la funzione sft_init, la quale verrà chiamata all'interno di proc_bootstrap.

- **runprogram:** La funzione runprogram è stata modificata per ricevere anche gli argomenti da linea di comando. Il nuovo prototipo della funzione contiene oltre al nome del programma anche il numero di argomenti e il vettore di puntatori a strighe. Queste verranno copiate nello stack e poi si procederà a chiamare la enter_new process con i corretti valori di argc e argv.
- **menu.c:** Si è modificata la funzione cmd_progthread adattandola alle nuove funzionalità (gestione degli argomenti e sys_waitpid in common_prog). Alla funzione common_prog è stata aggiunta la gestione dell'errore ENPROC dovuta alla presenza della process table. Inoltre è stata aggiunta la chiamata alla sys_waitpid per poter mettere il menu in attesa della terminazione del nuovo processo.
- **file_syscalls.c std_open:** Alla creazione di un processo è necessario gestire l'apertura dei file descriptor relativi a STDIN, STDOUT e STDERR. I tre file possono essere ottenuti effettuando una open sul path "con:". Non essendo una syscall vera e propria si è deciso di implementarla come funzione separata, con funzionamento analogo alla sys_open, ma specifica per il path della console. Alla creazione di ogni processo vengono effettuate tre chiamate a tale funzione per aprire i file descriptor standard. L'apertura di tali file è implementata tramite vfs_open, in modo analogo a una sys_open.
- **trap.c kill_curthread:** Nella versione iniziale di killcurthread non è presente una effettiva gestione dell'uscita del thread e del codice di uscita. In os161, a differenza che in Linux, non viene effettuato uno shift di 8 bit per distinguere i diversi tipi di valori di ritorno; si effettua invece uno shift di 2 bit, inserendo negli ultimi due bit i seguenti valori per distinguere uscite differenti (operazioni effettuate da macro definite in kern/wait.h):

  - 0: uscita con chiamata a exit()
  - 1: uscita in seguito a fatal signal
  - 2: uscita in seguito a fatal signal con dumped core
  - 3: processo stoppato senza uscita

  Si è pertanto integrata alla gestione iniziale del valore del segnale, presente nella killcurthread, la gestione del codice di uscita e la chiusura del thread.



4 Appendice

4.1 Demo di esecuzione

Si allega una demo di esecuzione della shell: `https://youtu.be/gNs7pqFE09o`

4.2 Repository github

`https://github.com/ciaccogiuseppe/os161-project-c2`

4.3 Errori non implementati (da Linux)

- **ENOSYS:** nella fork(), non implementata in quanto esterna allo scopo del progetto
- **EAGAIN:**
  - read/write: socket non gestiti
  - fork: EAGAIN in Linux corrisponde a ENPROC e EMPROC in OS161
  - execv: utenti non gestiti nell'implementazione attuale
- **EINTR:** in open, close, write, read, waitpid, dup2; interruzioni dovute a segnali non gestite nell'implementazione corrente
- **EINVAL:** per la dup2 in Linux è riferita alla concorrenza con dup3
- **EPERM:** in open, write, execv; gestione dei permessi non presente nell'implementazione corrente
- **EACCES:** in open, getcwd, chdir, execvM gestione dei permessi non presente nell'implementazione corrente
- **ELOOP:** in open, chdir, execv; link simbolici non gestiti a livello di emufs nell'implementazione corrente
- **ENXIO:** per la lseek, relativo a opzioni non presenti nell'implementazione corrente
- **EBUSY:** in open e dup2 non implementata in quanto esterna allo scopo del progetto
- **EMFILE:** in execv; tabella dei file non modificata da execv
- **ENFILE:** in execv; tabella dei file non modificata da execv
- **EPIPE:** in write; pipe non implementate
- **EROFS:** in open; modalità di accesso non presenti nell'implementazione corrente
- **ENOSPC:** in close; errore relativo a file system NFS in linux
- **EDQUOT:** in open, close, write; utenti non gestiti nell'implementazione attuale
- **EFBIG:** in open
- **EOPNOTSUPP:** in open; relativo a opzione non presente nell'implementazione corrente
- **EOVERFLOW:** in open (analogo a EFBIG); in fstat (non implementato in emufs)
- **ETXTBSY:** in open
- **EWOULDBLOCK:** in open, read, write; socket non gestiti nell'implementazione corrente e/o fcntl non implementata
- **ERESTARTNOINTR:** in fork; interruzioni dovute a segnali non gestite nell'implementazione corrente
- **ELIBBAD:** in execv

4.4 Report test os161

- **fortkest:** utilizzato per verificare il corretto funzionamento della syscall fork

```
testbin/forktest: Starting. Expect this many:
|--------------------------|
AABBBBCCCCCCCCDDDDDDDDDDDDDDDD
testbin/forktest: Complete.
```

- **bigexec:** utlizzato per verificare il corretto funzionamento della gestione dei parametri nella execv

```
testbin/bigexec: Starting.
testbin/bigexec: 1. Execing with one 8-letter word.
/testbin/bigexec: 2. Execing with one 4050-letter word.
/testbin/bigexec: 3. Execing with two 4050-letter words.
/testbin/bigexec: 4. Execing with 16 4050-letter words.
/testbin/bigexec: 5. Execing with one 16320-letter word.
/testbin/bigexec: 6. Execing with two 16320-letter words.
/testbin/bigexec: 7. Execing with four 16320-letter words.
/testbin/bigexec: 8. Execing with one 65500-letter word.
/testbin/bigexec: 9. Execing with 300 8-letter words.
/testbin/bigexec: 10. Execing with 3850 8-letter words.
/testbin/bigexec: Complete.
```

- **argtest:** utilizzato per verificare la corretta gestione dei parametri nella execv

```
p testbin/argtest a bb cc
argc: 4
argv[0]: testbin/argtest
argv[1]: a
argv[2]: bb
argv[3]: cc
argv[4]: [NULL]
```

- **bigfork:** utilizzato per verificare il corretto funzionamento della execv
- **bigseek:** utilizzato per verificare la corretta gestione del valore di ritorno nella lseek

```
Creating file...
Writing something at offset 0
Seeking to (and near) 0x1000
Writing something else
Seeking to (and near) 0x0
Checking what we wrote
Seeking to (and near) 0x1000
Checking the other thing we wrote
Seeking to (and near) 0x20
Seeking to (and near) 0x7fffffff
Seeking to (and near) 0x80000000
Seeking to (and near) 0x80000020
Seeking to (and near) 0x100000000
```

```
Seeking to (and near) 0x100000020
Seeking to (and near) 0x180000000
Seeking to (and near) 0x180000020
Now trying to read (should get EOF)
Now trying to write (should get EFBIG)
Seeking to (and near) 0x100000000
Trying to read again (should get EOF)
Passed.
```

- **badcall:** i test badcall sono stati utilizzati per verificare il corretto funzionamento della gestione degli errori nelle syscall implementate

```
OPEN
------------------------------------------------------------------------------
badcall: open with NULL path... Bad memory reference                    passed
badcall: open with invalid-pointer path... Bad memory reference         passed
badcall: open with kernel-pointer path... Bad memory reference          passed
badcall: open null: with bad flags... Invalid argument                  passed
badcall: open empty string... Invalid argument                          passed

READ
------------------------------------------------------------------------------
badcall: read using fd -1... Bad file number                            passed
badcall: read using fd -5... Bad file number                            passed
badcall: read using closed fd... Bad file number                        passed
badcall: read using impossible fd... Bad file number                    passed
badcall: read using fd OPEN_MAX... Bad file number                      passed
badcall: read using fd opened write-only... Bad file number             passed
badcall: read with NULL buffer... Bad memory reference                  passed
badcall: read with invalid buffer... Bad memory reference               passed
badcall: read with kernel-space buffer... Bad memory reference          passed

WRITE
------------------------------------------------------------------------------
badcall: write using fd -1... Bad file number                           passed
badcall: write using fd -5... Bad file number                           passed
badcall: write using closed fd... Bad file number                       passed
badcall: write using impossible fd... Bad file number                   passed
badcall: write using fd OPEN_MAX... Bad file number                     passed
badcall: write using fd opened read-only... Bad file number             passed
badcall: write with NULL buffer... Bad memory reference                 passed
badcall: write with invalid buffer... Bad memory reference              passed
badcall: write with kernel-space buffer... Bad memory reference         passed

LSEEK
------------------------------------------------------------------------------
badcall: lseek using fd -1... Bad file number                           passed
badcall: lseek using fd -5... Bad file number                           passed
badcall: lseek using closed fd... Bad file number                       passed
badcall: lseek using impossible fd... Bad file number                   passed
```

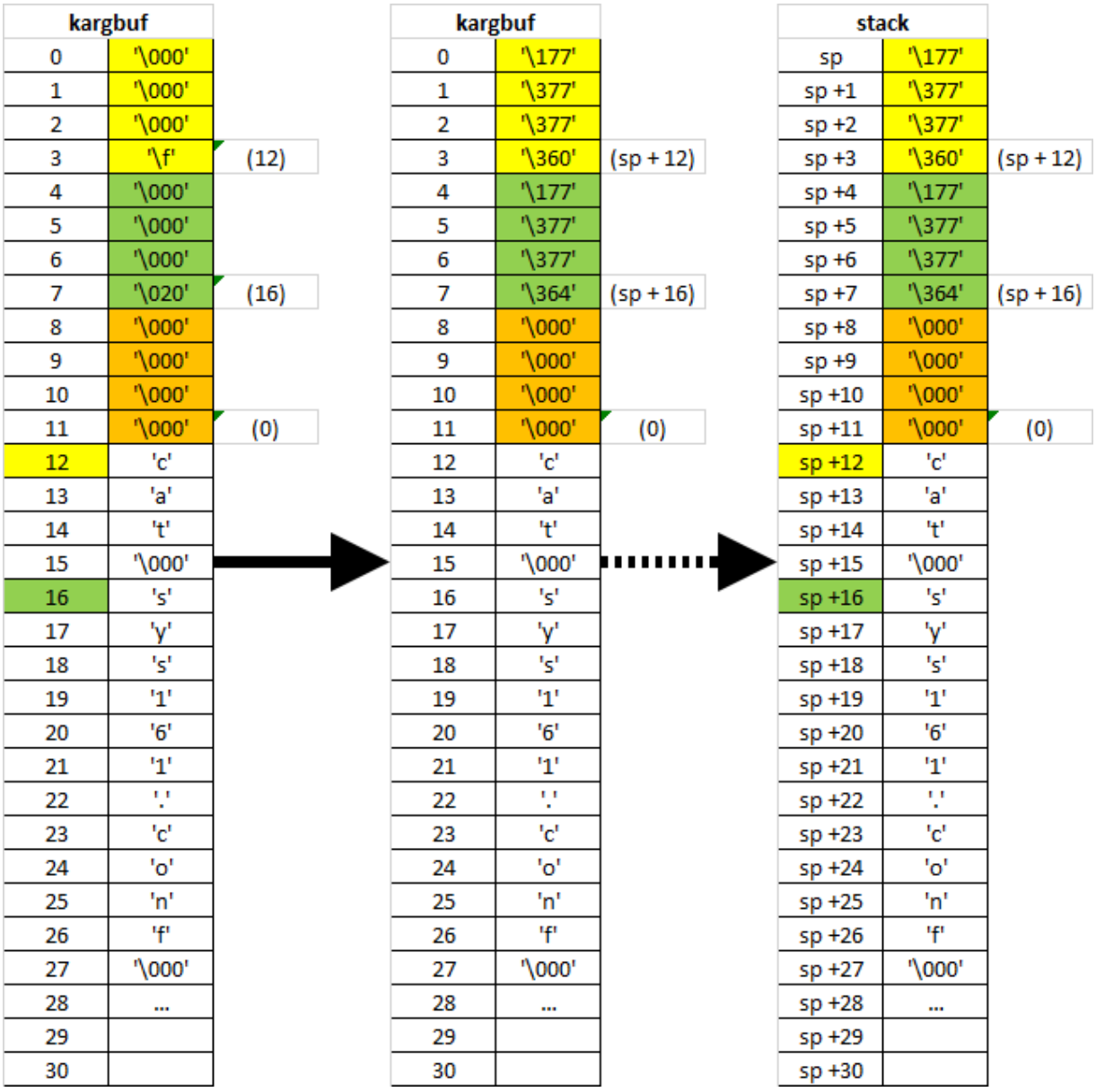
```
badcall: lseek using fd OPEN_MAX... Bad file number                    passed
badcall: lseek on device... Illegal seek                               passed
badcall: lseek stdin when open on file...                              ------
badcall: try 1: SEEK_SET... Success                                    passed
badcall: try 2: SEEK_END... Success                                    passed
badcall: lseek to negative offset... Invalid argument                  passed
badcall: seek past/to EOF...                                           passed
badcall: lseek with invalid whence code... Invalid argument            passed

CLOSE
-------------------------------------------------------------------------------
badcall: close using fd -1... Bad file number                          passed
badcall: close using fd -5... Bad file number                          passed
badcall: close using closed fd... Bad file number                      passed
badcall: close using impossible fd... Bad file number                  passed
badcall: close using fd OPEN_MAX... Bad file number                    passed

DUP2
-------------------------------------------------------------------------------
badcall: dup2 using fd -1... Bad file number                           passed
badcall: dup2 using fd -5... Bad file number                           passed
badcall: dup2 using closed fd... Bad file number                       passed
badcall: dup2 using impossible fd... Bad file number                   passed
badcall: dup2 using fd OPEN_MAX... Bad file number                     passed
badcall: dup2 to -1... Bad file number                                 passed
badcall: dup2 to -5... Bad file number                                 passed
badcall: dup2 to impossible fd... Bad file number                      passed
badcall: dup2 to OPEN_MAX... Bad file number                           passed
badcall: copying stdin to test with... badcall: dup2 to same fd...     passed
badcall: fstat fd after dup2 to itself... Success                      passed

CHDIR
-------------------------------------------------------------------------------
badcall: chdir with NULL path... Bad memory reference                  passed
badcall: chdir with invalid-pointer path... Bad memory reference       passed
badcall: chdir with kernel-pointer path... Bad memory reference        passed
badcall: chdir to empty string... Invalid argument                     passed

__GETCWD
-------------------------------------------------------------------------------
badcall: getcwd with NULL buffer... Bad memory reference               passed
badcall: getcwd with invalid buffer... Bad memory reference            passed
badcall: getcwd with kernel-space buffer... Bad memory reference       passed

EXECV
-------------------------------------------------------------------------------
badcall: exec with NULL program... Bad memory reference                passed
badcall: exec with invalid pointer program... Bad memory reference     passed
badcall: exec with kernel pointer program... Bad memory reference      passed
badcall: exec the empty string... Invalid argument                     passed
```

```
badcall: exec with NULL arglist... Bad memory reference                 passed
badcall: exec with invalid pointer arglist... Bad memory reference      passed
badcall: exec with kernel pointer arglist... Bad memory reference       passed
badcall: exec with invalid pointer arg... Bad memory reference          passed
badcall: exec with kernel pointer arg... Bad memory reference           passed

WAITPID
-------------------------------------------------------------------------------
badcall: wait for pid -8... No such process                             passed
badcall: wait for pid -1... No such process                             passed
badcall: pid zero... No such process                                    passed
badcall: nonexistent pid... No such process                             passed
badcall: wait with NULL status... Success                               passed
badcall: wait with invalid pointer status... Bad memory reference       passed
badcall: wait with kernel pointer status... Bad memory reference        passed
badcall: wait with unaligned status... Bad memory reference             passed
badcall: wait with bad flags... Invalid argument                        passed
badcall: wait for self... No child processes                            passed
badcall: wait for parent...
from child:              No child processes                             passed
from parent:             Success                                        passed
badcall: siblings wait for each other...
sibling (pid 9)                       No child processes                passed
sibling (pid 10)                      No child processes                passed
overall                                                                 passed

FSTAT
-------------------------------------------------------------------------------
badcall: fstat using fd -1... Bad file number                           passed
badcall: fstat using fd -5... Bad file number                           passed
badcall: fstat using closed fd... Bad file number                       passed
badcall: fstat using impossible fd... Bad file number                   passed
badcall: fstat using fd OPEN_MAX... Bad file number                     passed
badcall: fstat with NULL buf... Bad memory reference                    passed
badcall: fstat with invalid pointer buf... Bad memory reference         passed
badcall: fstat with kernel pointer buf... Bad memory reference          passed

GETDIRENTRY
-------------------------------------------------------------------------------
badcall: getdirentry using fd -1... Bad file number                     passed
badcall: getdirentry using fd -5... Bad file number                     passed
badcall: getdirentry using closed fd... Bad file number                 passed
badcall: getdirentry using impossible fd... Bad file number             passed
badcall: getdirentry using fd OPEN_MAX... Bad file number               passed
badcall: getdirentry using fd opened write-only... Bad file number      passed
badcall: getdirentry with NULL buffer... Bad memory reference           passed
badcall: getdirentry with invalid buffer... Bad memory reference        passed
badcall: getdirentry with kernel-space buffer... Bad memory reference   passed
```

4.5 Gestione dei parametri nella execv()

Nella figura sottostante si presenta un esempio di funzionamento della gestione dei parametri durante l'esecuzione della execv. L'esempio è relativo all'esecuzione del comando "cat sys161.conf" in bin/sh

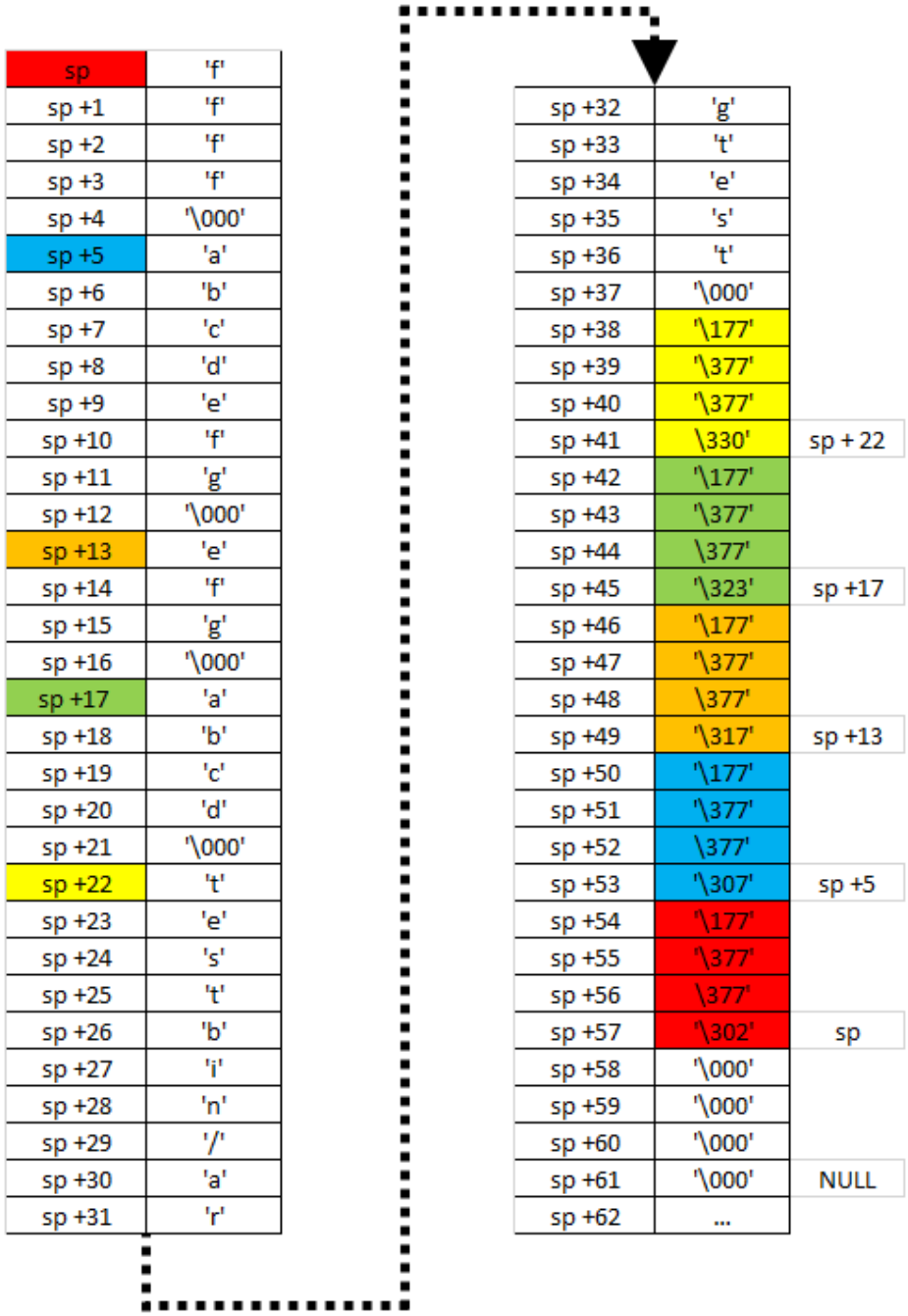
- A sinistra (**kargbuf**): contenuto di kargbuf dopo l'esecuzione di copy_args; al posto dei puntatori si hanno gli indici delle stringhe all'interno del vettore
- Al centro (**kargbuf**): contenuto di kargbuf dopo l'esecuzione di adjust_kargbuf: si hanno gli indirizzi che le stringhe avranno una volta copiate nello stack user
- A destra (**stack**): contenuto dello stack dopo la copia di kargbuf



Gestione parametri execv; l'allineamento dei puntatori è su 32 bit; il valore di sp è **0x7FFFFFE4**

4.6 Gestione dei parametri nella runprogram()

Nella figura sottostante si presenta un esempio di funzionamento della gestione dei parametri durante l'esecuzione di "p testbin/argtest abcd efg abcdefg ffff" partendo dal menu. Si rappresenta lo stack in seguito alla copia dei parametri in esso.



Gestione parametri runprogram; l'allineamento dei puntatori è su 32 bit; il valore di sp è **0x7FFFFFC2**; il valore di argvptr è **0x7FFFFFE8** (sp+38)