Linux Process & `` Workshop – Summary Sheet

Exercise 1: Forking a Child Process

Objective: Understand process creation using `fork()`. **Key Concepts:**

- `fork()` creates a child process with a new PID.
- Parent and child run independently.
- `wait()` makes the parent wait until the child exits.

Flow:

1. Parent prints message.
2. Child prints its PID and parent PID.
3. Both sleep to allow observation.
4. Parent waits for child and prints when child exits.

Example Output:

```
[Parent] Created child with PID = 1234
[Child] Started. PID = 1234, PPID = 1233
[Child] Doing some work...
[Child] Exiting now.
[Parent] Child has exited.
```

Exercise 2: Executing a Program in Child

Objective: Replace child process with another program using `execl()`. **Key Concepts:**

- `execl()` replaces the current process image with a new one.
- Parent waits for the child to finish executing the new program.

Example Output:

```
[Child] PID = 1235, going to exec /bin/ls in 2 seconds...
Parent waiting for child...
Child execution complete.
```

Exercise 3: Attaching to a Process with `ptrace`

Objective: Learn how to attach to a running child process. **Key Concepts:**

- `ptrace(PTRACE_ATTACH, pid, ...)` attaches to a process.
- `raise(SIGSTOP)` in the child can pause execution for tracing.
- `PTRACE_DETACH` resumes child execution.

Example Output:

```
[Child] PID = 1236, stopping for parent to attach...
[Parent] Attaching to child PID 1236...
[Parent] Attached! Sleeping 2 seconds before detach...
[Parent] Detached from child.
[Child] Continuing execution and exiting.
```

Exercise 4: Reading Child Process Memory

Objective: Read memory from a child process using `ptrace`. **Key Concepts:**

- Use `PTRACE_PEEKDATA` to read memory from a child.
- Child prepares a buffer; parent reads its contents.

Example Output:

```
[Child] PID = 1237, buffer at 0x7ffd... contains: 'Hello, World!'
[Parent] Attached! Attempting to read child buffer...
[Parent] Read data from child buffer (first 8 bytes): 0x57202c6f6c6c6548
[Parent] Detached from child.
```

Exercise 5: Writing to Child Process Memory

Objective: Modify a child process’s memory from the parent. **Key Concepts:**

- `PTRACE_POKEDATA` writes memory to the child.
- Parent can modify buffers in the child while it is stopped.

Example Output:

```
[Child] PID = 1238, buffer at 0x7fff... contains: 'Original'
[Parent] Attaching to child PID 1238...
[Parent] Attached! Writing 'Modified' to child's buffer...
[Parent] Verified child buffer first 8 bytes: 0x6465696669646f4d
[Parent] Detached from child.
[Child] Buffer after modification: 'Modified'
[Child] Exiting.
```

Exercise 6: Simplified Process Injection

Objective: Demonstrate memory injection into a child process using `ptrace`. **Key Concepts:**

- Child stops itself with `raise(SIGSTOP)` to allow parent injection.
- Parent writes new data into child's buffer using `PTRACE_POKEDATA`.
- Child prints buffer after injection to verify modification.

Example Output:

```
[Child] PID = 1239, buffer at 0x7ffd... contains: 'Original data'
[Parent] Attached! Injecting 'Injected data' into child's buffer...
[Parent] Injection complete and detached.
[Child] Buffer after injection: 'Injected data'
[Child] Exiting.
```

General Notes for Students

- Always check `pid` values — parent and child will have different PIDs.
- `ptrace` may require **root privileges** or `ptrace_scope=0` on modern Linux.
- Buffers are read/written in chunks of `sizeof(long)` for memory alignment.
- Using `raise(SIGSTOP)` is useful for pausing the child to allow inspection or monitoring.