41147011S 鄭聿喬 資工115

學習與反思報告

主題：基於 UNIX 信號處理和子程序執行的程式設計

學習內容與過程

在這次實作中，我開發了一個基於 UNIX 系統的 C 程式，該程式可以在接收到 `CTRL-C` 信號（`SIGINT`）或每隔 10 秒時，執行兩個系統命令 `uptime` 和 `who`。整體過程涉及以下技術點：

1. **信號處理**
   使用 `signal()` 函數攔截 `SIGINT` 信號（通常由 `CTRL-C` 觸發），並調用自定義的信號處理函數 `handle_sigint()`。這是 UNIX 系統中進行異步事件處理的核心技術，使得程式能夠在用戶觸發的情況下執行特定操作。

2. **子程序管理**
   使用 `fork()` 函數創建子進程，並透過 `execlp()` 執行系統命令。每個子程序完成後，主程序使用 `wait()` 進行同步，以確保執行順序的正確性。

3. **程式控制**
   主程序設置了一個無窮迴圈，透過 `sleep(10)` 每隔 10 秒執行一次子程序，並與信號處理邏輯共同實現「定時執行」與「用戶中斷執行」的功能。

執行結果分析

透過測試，程式成功實現了以下功能：

1. **按下 `CTRL-C` 時觸發指令執行**
   每次按下 `CTRL-C`，程式會執行 `uptime` 和 `who` 命令，並正確顯示執行結果。這展示了信號處理機制的即時性和可靠性。

2. **定時執行功能**
   程式每隔 10 秒自動執行一次指令，符合預期設計，展示了程式對時間控制的處理能力。

3. **穩定的子程序管理**
   通過 `fork()` 與 `wait()` 的結合，程式能保證子進程按序執行，並在主進程等待完成後繼續運行，避免了多進程執行中的競爭問題。

4. **程式可持續運行**
   主程序在信號處理與定時執行邏輯的結合下，可以持續運行且不會崩潰，展示了良好的穩定性。

挑戰與解決

1. **信號處理的安全性問題**
   起初，信號處理函數中直接進行了 `fork()` 與指令執行，導致執行時偶爾出現不穩定現象。查閱文檔後得知，在信號處理函數中應避免執行複雜邏輯，因此將大部分邏輯抽取到獨立函數中。

2. **多進程執行順序問題**
   初始版本未使用 `wait()` 導致子進程的輸出順序不一致，經過調整後，通過 `wait()` 同步了子進程的執行，確保輸出順序正確。

3. **程式無法停止的問題**
   由於程式是無窮迴圈運行，測試時發現難以終止。解決方案是透過 `kill` 命令或者修改程式增加特定退出條件（例如檢測用戶輸入 `q`）。

學習收穫

1. **信號處理的應用**
   通過這次實作，我對 UNIX 信號處理有了更加深入的了解，學會了如何攔截特定信號並執行相應的動作。

2. **進程管理的能力提升**
   我學會了如何使用 `fork()` 、 `execlp()` 和 `wait()` 來實現多進程程序的協同運行，並對多進程程序的執行順序進行控制。

3. **問題排查與解決**
   通過處理信號安全性問題與多進程的競爭問題，我掌握了閱讀文檔和分析問題的能力。

反思與未來改進

1. **程式易用性**
   當前程式只能通過外部命令停止，對用戶不太友好。未來可以考慮加入檢測特定輸入（例如按 `q` 退出）的功能，提升用戶體驗。

2. **多進程的資源管理**
   現在的程式對子進程的錯誤處理較少，例如如果子進程執行命令失敗，主進程無法進一步處理。未來可以增加錯誤日誌和自動重試機制。

3. **功能擴展**
   此程式目前僅執行 `uptime` 和 `who` 指令，未來可以設計更加通用的框架，允許用戶指定需要執行的命令集，實現更靈活的功能。

結語

這次實作不僅讓我鞏固了 UNIX 系統的基礎知識，還提升了我對多進程程序設計的理解與實踐能力。透過解決實際問題，我對系統調用與程式運行的細節有了更深的體會，這些經驗將對我未來的開發工作大有裨益。