Wróćmy teraz na chwilę do zagadnienia sygnałów i komunikacji międzyprocesowej. Spróbujemy napisać kilka skryptów, które pokażą jak w działają sygnały na przykładzie skryptów, ale podobne mechanizmy istnieją w C.

Zacznijmy od tego jak pisać funkcje w skryptach shellowych. Funkcje pozwalają wydzielić częściej wykonywane fragmenty kodu. Czynią w ten sposób program bardziej czytelnym i łatwiejszym w poprawianiu. W shellu deklaracja funkcji zaczyna się od jej identyfikatora (nazwy), pary nawiasów () i w nawiasach klamrowych {} umieszczamy kod funkcji:

```
hello () {
    echo "Hello!"
}
```

Do funkcji możemy przekazywać parametry pozycyjne dokładnie tak jak do skryptu shellowego: od $1 do $9.

Do przechwycenia sygnału w shellu służy funkcja `trap`. Woła się ją podając przynajmniej dwa parametry: identyfikator funkcji obsługującej sygnał oraz nazwy symboliczne lub numeryczne skojarzonych sygnałów. Przywołajmy skrypt, który już widzieliśmy wcześniej z drobną zmianą:

```
#!/bin/sh

hello () {
    echo "Hello!"
}

surprise () {
    echo "Surprise!"
}

trykill () {
    echo "Trying to kill"
}

trystop () {
    echo "Trying to stop"
}

trap hello INT
trap surprise ALRM
trap trykill KILL
trap trystop STOP

echo "My PID is:" $$

i=0
while true
do
    echo $i
    sleep 1
    i=`expr $i + 1`
done
```

Na początku zadeklarowane są 4 funkcje: `hello()`, `surprise()`, `trykill()`, oraz `trystop()`. Następnie wołamy 4 razy `trap()` aby skojarzyć sygnały z naszymi funkcjami. Skrypt na początku wypisuje swój identyfikator procesu, który jest przechowywany w specjalnej zmiennej `$$`. Ułatwi to

nam przesyłanie do niego sygnałów. Potem wykonywana jest nieskończona pętla `while`. Inicjujemy licznik `i` na 0. Warunek w pętli `while` jest zawsze prawdziwy, dlatego to pętla nieskończona. W pętli wypisujemy wartość licznika `i`, odczekujemy (śpimy) 1 sekundę, następnie zwiększamy wartość licznika `i` o 1. Ponieważ w shellu nie ma operacji arytmetycznych, wykorzystujemy do tego celu program `expr`. Jego wykonanie ujęte jest w lewe apostrofy, więc wynik wstawiany jest do zmiennej `i`. Zatem nasz skrypt po uruchomieniu wypisze swój PID i co sekundę będzie wypisywał kolejne liczby naturalne od 1.

Naciśnięcie CTRL+C zwykle przerywa program. Tutaj sygnał `SIGINT`, czyli ten który wysyłany jest przez CTRL+C, skojarzony jest z funkcją `hello()`. Dlatego po naciśnięciu CTRL+C zobaczymy napis `Hello!`, a skrypt wykonywany będzie dalej. Aby do naszego skryptu wysłać sygnał SIGALRM musimy otworzyć nowe okno terminala. Tam wpisujemy `kill -ALRM PID`, gdzie PID to identyfikator procesu naszego skryptu. W oknie skryptu powinniśmy zobaczyć napis `Surprise!`. Spróbujmy teraz wysłać sygnał `SIGKILL` czyli 9. Nie zobaczymy `Trying to kill` bo jak pamiętamy tego sygnału nie można oprogramować. Skrypt zostanie przerwany. Po ponownym uruchomieniu wyślijmy sygnał `SIGSTOP`. Tego sygnału także nie można przechwycić. Wykonanie skryptu zostanie wstrzymane bez wypisania `Trying to stop`. Możemy to sprawdzić poprzez `jobs` w jego terminalu.

Zobaczmy do czego bardziej praktycznego można użyć sygnałów. Nie jest to przykład z życia, ale pokazuje, co sensownego można uzyskać za pomocą sygnałów.

```
#!/bin/sh

TEMP_FILE=/tmp/temp_file.txt
THIS_IS_THE_END=0

surprise () {
    echo "Got SIGNAL, but I'm still alive" >> $TEMP_FILE
}

break_loop () {
    THIS_IS_THE_END=1
}

cleanup () {
    rm -f $TEMP_FILE
    exit
}

trap surprise 2
trap cleanup 1 15
trap break_loop 19

echo "My PID is:" $$

i=0
while true
do
    echo $i >> $TEMP_FILE
    if [ $THIS_IS_THE_END -eq 1 ]
    then
        break
    fi
    sleep 1
    i=`expr $i + 1`
done
```

```
echo "Wydruk [t/n]: "
read r
if [ "$r" = "t" ]; then
    cat $TEMP_FILE
fi

cleanup
```

Na samym początku inicjujemy dwie zmienne. W `$TEMP_FILE` umieszczona jest nazwa pliku tymczasowego, w którym zapisywane będą wyniki. Zmienna `$THIS_IS_THE_END` sprawdzana jest w głównej pętli programu i służy do jej przerwania. Dalej mamy zadeklarowane trzy własne funkcje: `surprise()`, `break_loop()` oraz `cleanup()`. Pierwsza dopisuje do pliku informację o odebraniu sygnału, druga ustawia zmienną `$THIS_IS_THE_END`, co spowoduje przerwanie głównej pętli programu. Trzecia usuwa plik tymczasowy i powoduje zakończenie programu. Ta funkcja jest tutaj istotna. Zwykle aplikacje w trakcie działania używają wielu plików tymczasowych do przechowywania danych, a przy wyjściu usuwają swoje pliki tymczasowe by nie zajmowały miejsca. W tym celu przechwytywany jest przez aplikację sygnał `SIGTERM` normalnie otrzymywany na przykład przy zamykaniu okna programu. Przy zamknięciu terminala, w którym wykonywany jest skrypt, do skryptu wysyłany jest sygnał `SIGHUP`.

Sygnał `SIGINT` obsługiwany jest przez funkcję `surprise()`, sygnały `SIGHUP` i `SIGTERM` obsługiwane są przez funkcję `cleanup()`, natomiast sygnał `SIGPWR` przez `break_loop()`.

Podobnie jak poprzednio na początku wypisywany jest PID skryptu, a pętla główna `while` w nieskończoność wypisuje kolejne liczby naturalne w odstępach co sekundę. Różnica polega na tym, że liczby dopisywane są do pliku tymczasowego, nie na standardowe wyjście. Dodatkowo w instrukcji warunkowej if sprawdzana jest wartość zmiennej `$THIS_IS_THE_END` i jeśli ma ona wartość równą 1 to pętla jest przerywana.

Jeśli pętla zostanie przerwana przez sygnał `SIGPWR` to zostaniemy zapytani, czy wypisać zawartość pliku tymczasowego. Odpowiedź z klawiatury czytana jest przez funkcję `read()` i wstawiana do zmiennej `r`. Jeśli ma ona wartość `t` to wypisana zostanie zawartość pliku tymczasowego. Na koniec plik tymczasowy jest usuwany.

Po odebraniu sygnału `SIGHUP` lub `SIGTERM` wykonywana jest funkcja `cleanup()`. Plik tymczasowy jest usuwany, a program jest kończony. W trakcie działania skryptu, w innym oknie terminala możemy obejrzeć, co jest w pliku tymczasowym:

```
cat /tmp/temp_file.txt
```

o ile nie zmieniliśmy wartości zmiennej `$TEMP_FILE`. Po zamknięciu okna z uruchomionym skryptem plik tymczasowy powinien zniknąć.