Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

KIV/OS Operační systémy

Single-task výpočet

Jan Hinterholzinger

A22N0045P
hintik@students.zcu.cz

2. 12. 2022 Plzeň

Obsah

1 Zadání

Motivace: Nízkopříkonová zařízení občas musí vykonat netriviální výpočet v rozumně omezeném čase, jehož výsledek je dostatečně dobrý (např. z pohledu bezpečnosti, přesnosti, ...). V tomto zadání implementujete patřičné prostředky k realizaci takového výpočtu v rámci jednoho tasku.

- realizováno na platformě Raspberry Pi Zero
- vytvoříte právě jeden user-space task, který bude zaštiťovat níže popsaný výpočet
- task po spuštění vypíše na UART základní informace o práci (číslo zadání a název, jméno řešitele a os. číslo) a napíše, co očekává za vstup
- task čte z UARTu parametry a vstupní data v textové podobě, každé zadání je ukočeno zalomením řádku (dle platformy)
- jako první vstup bude celé číslo, které označuje časový rozestup t_delta hodnot časové řady v minutách - tedy vlastně po jakých číslech se krokuje na ose X
- jako další vstup bude celé číslo, které označuje predikční okénko t_pred - tedy parametr modelu, viz níže
- jako každý další vstup bude číslo s plovoucí desetinnou tečkou nebo příkaz "stop"nebo "parameters"
- task provede po každém přijatém čísle fitting modelu

  $$y(t + t\_pred) = A * b(t) + B * b(t) * (b(t) - y(t)) + C$$

  , kde $b(t)$ spočítáte jako:

  $$b(t) = D/E * dy(t)/dt + 1/E * y(t)$$

- jako odpověď na přijaté číslo task vypíše novou predikci y(t + t_pred) dle nově spočtených parametrů, popř. řetězec NaN, pokud ještě není možné predikovat
- UART terminál musí být během výpočtu rozumně responzivní kdykoliv probíhá výpočet, musí ho být možné zastavit zadáním příkazu "stop" (bez uvozovek)

- pokud je výpočet ukončen, predikce je spočtena dle starých parametrů
- příkaz "parameters"vypíše na UART hodnoty parametrů A až E v nějaké rozumné formě
- pokud zrovna neprobíhá výpočet, zařízení minimalizuje spotřebu el. energie např. instrukcí WFI
- vstupy jsou pochopitelně rozumně validovány, pokud je zadán nevalidní vstup, task uživatele patřičně upozorní

2 Analýza úlohy

Zadání vyžaduje implementaci operačního systému na platformě Raspberry PI Zero. Jedná se o embedded zařízení, které je typicky určeno k jednoúčelovému, ale specifickému nasazení.

Pro implementaci budeme vycházet ze základu tvořeného na cvičeních z projektu KIV-RTOS, kde je již řada funkcí implementována. Cílem tedy je z projektu využít již implementované funkcionality a nepotřebné odstranit nebo ponechat nevyužité.

Následně je potřeba implementovat v uživatelském procesu vlastní logiku aplikačního procesu, který zaobstarává generování predikovaných hodnot na základě matematického modelu.

2.1 Využití funkcionalit RTOS

Jak již bylo zmíněno, pro implementaci byl zvolen výchozí bod konečný projekt ze cvičení RTOS. Poskytuje totiž dostatečnou konstru pro další rozšiřování a nabízí již implementované funkce.

- Základní správa paměti jádra
- Základní filesystém s propojením na různé vstupy (UART, TRNG)
- Plánování procesu (použito pro přepnutí z jádra do user procesu)
- Přerušení

Avšak je potřeba některé potřebné funkce doplnit.

2.1.1 Správa paměti

Primitivní správa paměti kernelu je již implementována v základním RTOS. Je potřeba ji však vylepšit algoritmem pro odstraňování děr.

Také je nutné implementovat přiřazování paměti procesu resp. haldy procesu. Pro zjednodušení úlohy nebude potřeba implementovat komplexní přidělování paměti. Stačí nám pouze přidělení konstantní velikosti paměti procesu (např. jedna stránka), kterou si potom proces sám organizuje.

2.1.2 Buffer pro UART

UART má svůj interní buffer o velikosti 8B. To je ovšem poměrně málo a pravděpodobně se stane, že nám nebude stačit. Abychom zamezili přepisování dat, tak je potřeba z tohoto bufferu data vyzvedávat a ukládat je do nějaké větší struktury.

Vytvoříme tedy buffer (nejlépe kruhový, aby při zaplnění pokračoval v plnění na začátku) o větší velikosti (např. nižší stovky bajtů). Poté například pomocí přerušení, vyvolaného při vstupu dat na UART, se data přesunou z malého bufferu do našeho většího bufferu. Z tohoto bufferu se již bude uživatelský proces číst příchozí data a zpracovávat je.

2.1.3 Implementace čtení z UART

V rámci základní implementace UART je dostupné vypisování skrz UART, nikoli však čtení. Proto je potřeba doimplementovat abstrakci čtení a následnou realizaci obsluhy v UART driveru.

V průběhu budeme pravděpodobně potřebovat dva typy čtení. Blokující pro zejména čtení zadávaných hodnot a parametrů a neblokující pro použití v momentě výpočtu pro zadání příkazu stop.

2.1.4 Úspora energie

Úspora energie je u embedded zařízení velmi důležitá specifikace. Taková zařízení jsou často napájeny z akumulátorů. Je zde tedy chtěný záměr prodloužit dobu výdrže zařízení při takovém omezeném zdroji. Jedním z nástrojů, kterým můžeme výdrž prodloužit jsou instrukce procesoru WFI a WFE, při kterých procesor pracuje v „omezeném režimu“. Procesor tak plně nepracuje a není tolik náročný na napájení a šetří energii.

Otázkou je, kde takové instrukce použít aniž by byla omezena funkčnost procesu. Takové místo v algoritmu existuje při čekání na vstup od uživatele. Pokud je tedy očekáván vstup od uživatele, tak se nic jiného nepočítá a jsou nasazeny zmíněné instrukce dokud uživatel vstup nezadá a nespustí tím nový výpočet.

2.2 Uživatelský proces

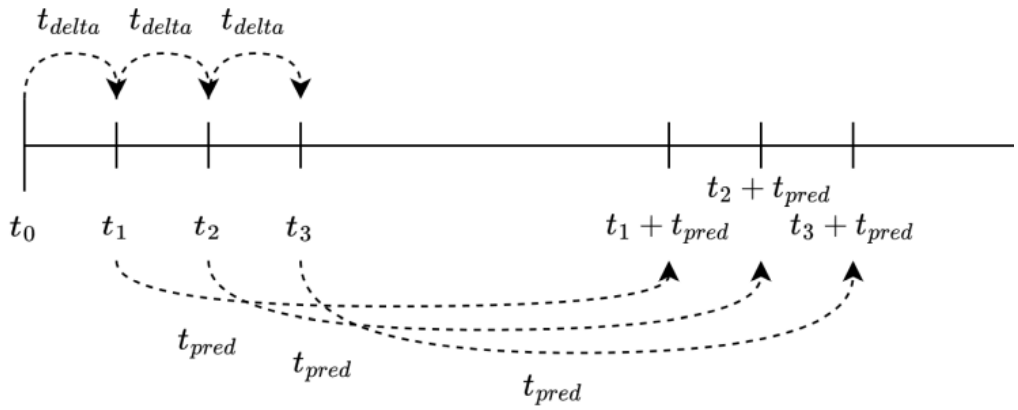
Samotný uživatelský proces již bude implementován podobně jako uživatelské procesy v jiných systémem. Samozřejmě však za pomocí vlastních knihoven a metod pro komunikaci.

Uživatelský program bude komunikovat pomocí rozhraní UART, přes které bude data vysílat i přijímat. Je tedy potřeba implementovat obousměrnou komunikaci na tomto rozhraní. Možností je několik, ale nejvýhodnější je využít abstrakci souborového systému, která již v RTOS je implementována.

2.2.1 Výpočet predikce

Výpočet predikce můžeme rozdělit na dvě části zjištění parametrů modelu (`A`, `B`, `C`, `D`, `E`) a samotný výpočet z modelu po dosazení zmíněných parametrů.

Ta druhá část je při znalosti parametrů elementární operace a jedná se pouze o dosazení do vzorce. Avšak zjištění těchto parametrů modelu je náročnější činnost. Je potřeba nalézt takové parametry vycházejících ze zadávaných dat `y(t)`, aby po dosazení byla predikce dostatečně přesná. Takový úkol se nazývá model fitting, neboli přizpůsobení modelu zadávaným datům.



Obrázek 1: Vztah mezi krokování a predikací

Možností jak zjistit parametry modelu je několik. Nejvíce však byly zvažovány následující přístupy:

- Lineární regrese – Metodou nejmenších čtverců proložení přímkou
- Polynomiální regrese – Metodou nejmenších čtverců proložení např. parabolou
- Numerické algoritmy
- Genetický algoritmus

Na základě složitosti algoritmů pro realizaci regrese (např. implementace reprezentace matic, Gaussova eliminační metoda, apod.) byla zvolena implementace pomocí genetických algoritmů. Tento přístup sice do výsledku zanese

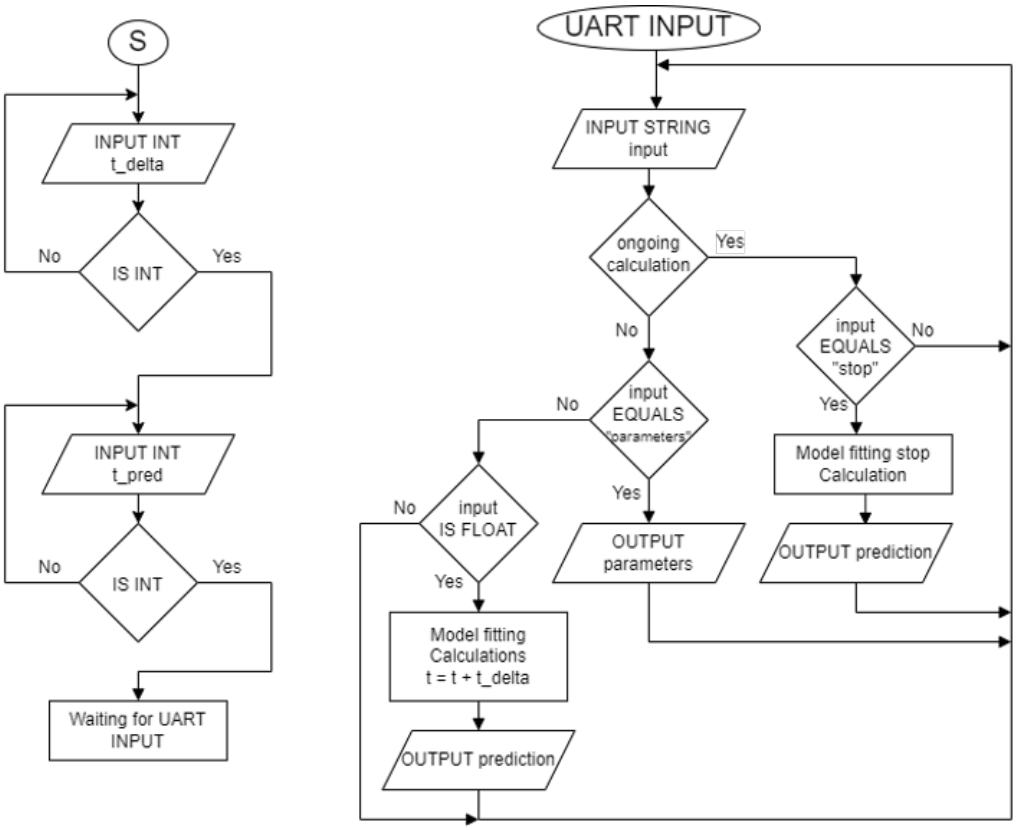
jistou míru náhody díky počáteční randomizaci. Avšak při dostatečné a různorodé mutaci populace je možnost získání dostatečně přesných výsledků.

2.2.2 Genetický algoritmus

Genetický algoritmus je druh evolučních algoritmů. Na počátku je stanovena (např. náhodně) množina předběžných výsledků. Tato množina se nazývá populace. je v každém cyklu modifikována různými způsoby – mutacemi (např. negace náhodného bitu, prohození částí výsledku, apod.). Mutování probíhá v cyklech, kdy po každém cyklu vznikne nová „generace“. Pro funkčnost algoritmu je stěžejní určit, které výsledky přejdou z jedné generace do druhé ideálně tak, aby bylo zajištěno vylepšování. K tomu slouží tzv. fitness funkce, která každý výsledek ohodnotí jak moc se blíží k optimálnímu řešení. Do dalšího cyklu přechází ta část populace s nejlepším ohodnocení fitness funkcí.

2.2.3 Algoritmus uživatelského procesu

Program získá z UART rozhraní dvě celočíselné hodnoty (`t_delta` – velikost krokování a `t_pred` – vzdálenost predikce). Následně program očekává vstup hodnot y_t. Poté co program nasbírá dostatečné množství dat a schopen stanovit predikci `y(t + t_pred)` (proces je schopen určit odchylku predikce od reálných dat) je po každém zadání hodnoty model (jeho parametry) přepočítán (provede se model fitting) a je predikována nová hodnota.



Obrázek 2: Hrubé schéma algoritmu uživatelského procesu

3 Popis implementace

3.1 Vyžití RTOS

Z většinové části byl systém implementován na základě existujícího projektu KIV-RTOS, který byl pouze doplněn o potřebné funkce.

User procesu je při spuštění automaticky přiřazena jedna stránka paměti. Pokud procesu tato stránka nestačí, kernel procesu alokuje stránku další.

3.1.1 Randomizace

v uživatelském procesu je randomizace implementována ve funkci `rand(uint32_t file, uint32_t output_start, uint32_t output_end)`. V této funkci je implementování získání náhodné hodnoty z registru `RNG` pomocí napojení na filesystém. Tato hodnota následně projde škálováním na požadovaný rozsah.

```
uint32_t trng_file = open("DEV:trng", NFile_Open_Mode::Read_Only);
uint32_t num = 0;
uint32_t output_end = 10; // max
uint32_t output_start = 5; // min
uint32_t input_end = 4294967295; //max uint
uint32_t input_start = 0; // min uint
//generovani nahodneho uint cisla
read(trng_file, reinterpret_cast<char*>(&num), sizeof(num));
double slope = 1.0 * (output_end - output_start) / (input_end - input_start);
uint32_t output = output_start + slope * (num - input_start);
```

Zdrojový kód 3.1: Randomizace z zadaném rozsahu

3.1.2 Výpočet predikce

Program po zadání hodnot velikosti krokování a predikce se ocitne v nekonečné smyčce, která blokovaně čte vstup z UART a následně dle něj větví program. Mohou nastat tyto možnosti vstupu:

- Prázdný vstup – program vyžaduje další vstup
- „stop“ – při blokujícím zadávání vstupu postrádá smysl – stejné chování jako při prázdném vstupu

- „parameters“ – program vypíše parametry modelu
- Číslo – program zahájí výpočtovou proceduru se zadaným číslem, příp. přepočte parametry modelu a příp. vypíše predikci
- Jiný vstup – Program oznámí neplatný vstup a očekává nový vstup

Při zahájení propočtu jsou provedeny mutace populace pro minimalizace fitness funkce. Jsou implementovány mutace křížením,

3.1.3 Výpočet fitness funkce

Fitness funkce je pro ulehčení implementována jako absolutní hodnota z rozdílu reálné a predikované hodnoty. To znamená, že při shodě predikované a reálné hodnoty je hodnota fitness funkce rovna nule. Obecně však pro hodnotu naší fitness funkce platí, že čím je hodnota nižší, tím je výsledek lepší.

4 Řešené problémy

4.1 Neblokující čtení

Pro větší responzivitu systému v době výpočtu bylo zamýšleno implementování neblokujícího čtení. Zejm. proto, aby bylo možné v průběhu výpočtu výpočet zastavit. To se však nepodařilo implementovat z důvodu nedařícího se propojení s přerušením.

4.2 Pokusy o nasazení na reálné zařízení

V závěru vývoje proběhlo několik pokusů o nasazení na zapůjčené zařízení. Avšak operační systém se na něm zprovoznit nepodařilo. Zařízení neprovádělo kód ani uživatelského procesu, ani jádra. Pravděpodobně se vyskytla chyba ještě dříve.

Pokusy o zprovoznění základního KIV-RTOS proběhly úspěšně a zařízení regovalo, z toho usuzuji že chyba je obsažena někde v pozdějších úpravách jádra.

Zprovoznění ani po několika úpravách nebyla úspěšná aniž by nebyla omezena funkčnost v simulátoru qemu, proto následně bylo upuštěno od realizace na reálném zařízení.

5 Testování

5.1 Emulované prostředí

Reálné zařízení bylo emulování pomocí emolátoru qemu. Na tento emulátor se následně připojoval debugger gdb, který umožňuje zobrazení registrů, zobrazení prováděného kódu aplikace, přidávání breakpointů a krokování.

5.2 Reálné zařízení

Reálné zařízení se testovalo obtížně. V počátku projektu typicky metodou pokus-omyl a signalizací pomocí ACT_LED a dalšími LED připojenými GPIO piny. V pozdější fázi organizovanými ladícími výpisy přes UART zachycované na PC v programu PuTTy.

Toto testování bylo však velmi omezené zejména při vývoji jádra, protože nebyla možnost jednoduše zobrazit obsahy jednotlivých registrů a spojit prováděné instrukce s kódem.

6 Závěr

Operační systém na emulovaném prostředí provádí kód a dává o sobě vědět přes rozhraní UART. Přijímá vstup a na základě něho provádí výpočty a vypisuje predikce na základě modelu. Na reálném zařízení se nepodařilo operační systém zprovoznit. Také se nepovedlo implementovat výpočet dostatečně responzivně, aby bylo možné během výpočtu zadat příkaz `stop` pro ukončení výpočtu. Výpočet je avšak dostatečně rychlý, že pro většinu případů pro to není potřeba.